RasPi

DESIGN
BUILD
CODE

39

Get hands-on with your Raspberry Pi

BUILD A NETWORKED HI-FI

+ 7 AMAZING PI PROJECTS

Plus Get data from your car with Pi

Welcome

Reggae legend Bob Marley was once quoted as saying, "One good thing about music, when it hits you, you feel no pain". While we'd hate to contradict the musical genius of Jamaica we'd wager he never had to endure "Let it Go" from Frozen 10 times in succession while on a car journey with his six year old daughter. No, probably not.

Instant access to good, and painless, music is the topic of our lead tutorial this issue where we show you how to put the Pimoroni pHAT DAC together with a Pi Zero to create a networked hi-fi. A high-quality networked hi-fi, no less, that takes advantage of the UK's online radio stations. Other highlights include a look at a Macintosh classic made from Lego and the final part of our tutorial on creating a Sense HAT battleships game.

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

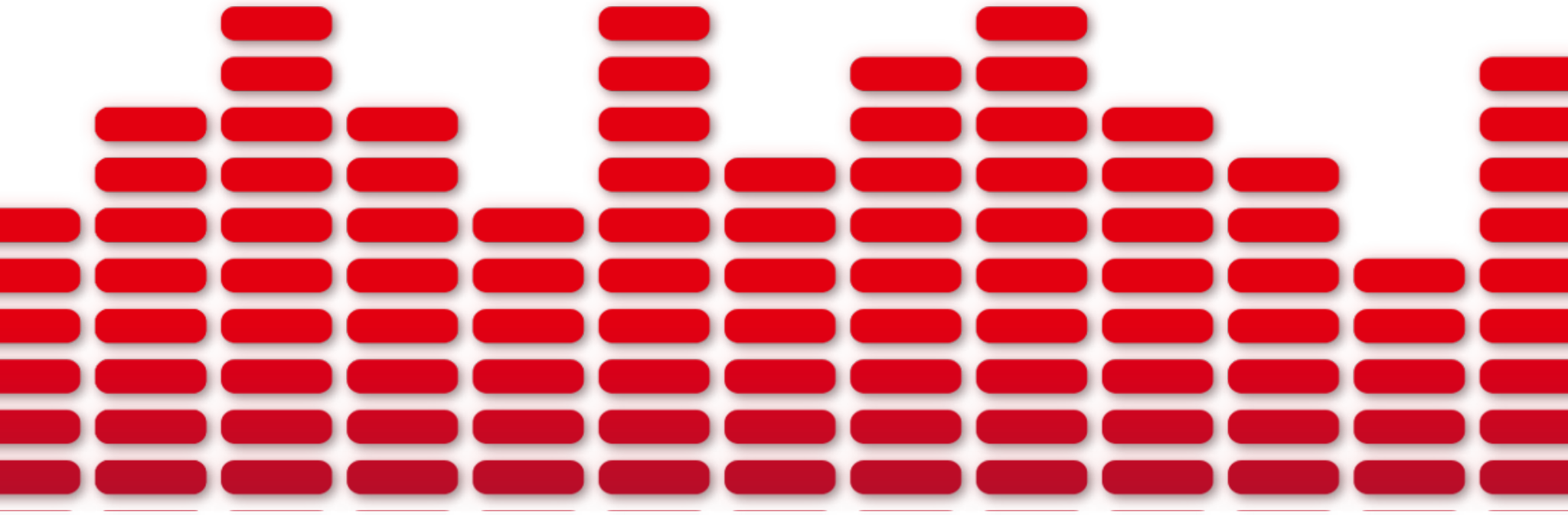
Editor

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com



Contents

Build your own networked hi-fi

Put the Pimoroni pHAT DAC together with a Pi Zero to create a networked hi-fi

THE PROJECT ESSENTIALS

Pimoroni pHAT DAC
Pi Zero

We will show you how to create a high-quality networked music player that takes advantage of the UK's online radio stations, Linux's popular Music Player Daemon, and a responsive web-server to control it all. The full-sized Raspberry Pis have two built-in audio outputs: audio over HDMI cable and a 3.5mm headphone jack that can suffer interference and noise. The Pi Zero itself has no audio jacks but Pimoroni has come to the rescue and built a high-quality DAC (digital audio converter) using the same chip as the Hi-Fi berry (PCM5102A).

01 Soldering the headers

The pHAT DAC comes with a 40-pin header, which you will need to solder. We consider a flux pen, work-lamp and thin gauge 60/40 solder essential for this. An optional RCA jack can also be bought fairly easily to give a phono-lead output for older stereos.

02 Install drivers

The DAC relies on I2C, so we have to load some additional kernel modules. If you are running Raspbian then

you can type in the following for a one-script installation over secure HTTP:

```
curl -sS https://get.pimoroni.com/phatdac | bash
```

While HTTPS provides a secure download, curious types may want to review the script before running it.

03 Installing Music Player Daemon (MPD)

Now install the MPD package and enable it to start on boot. MPD will be the backbone of the project providing playback of MP3s and internet radio stations. The MPC (client) software is also installed for debugging and setting up your initial music playlists:

```
sudo apt-get install mpd mpc
sudo systemctl enable mpd
```

04 Clone and install pyPlaylist web-server

pyPlaylist is a responsive (mobile-ready) web-server written with Python & Flask web framework. Once configured it will give us a way of controlling our Hi-Fi through a web-browser. The following will install pyPlaylist on Raspbian:

```
sudo pip install flask python-mpd2
cd ~
git clone https://github.com/alexellis/
pyPlaylist
cd pyPlaylist
./raspbian_install.sh
```

05 Choosing the radio stations

We put together a list of popular radio stations in the UK which can be run into MPD with the add_stations.sh file. You can find your own from radiofeeds.co.uk.

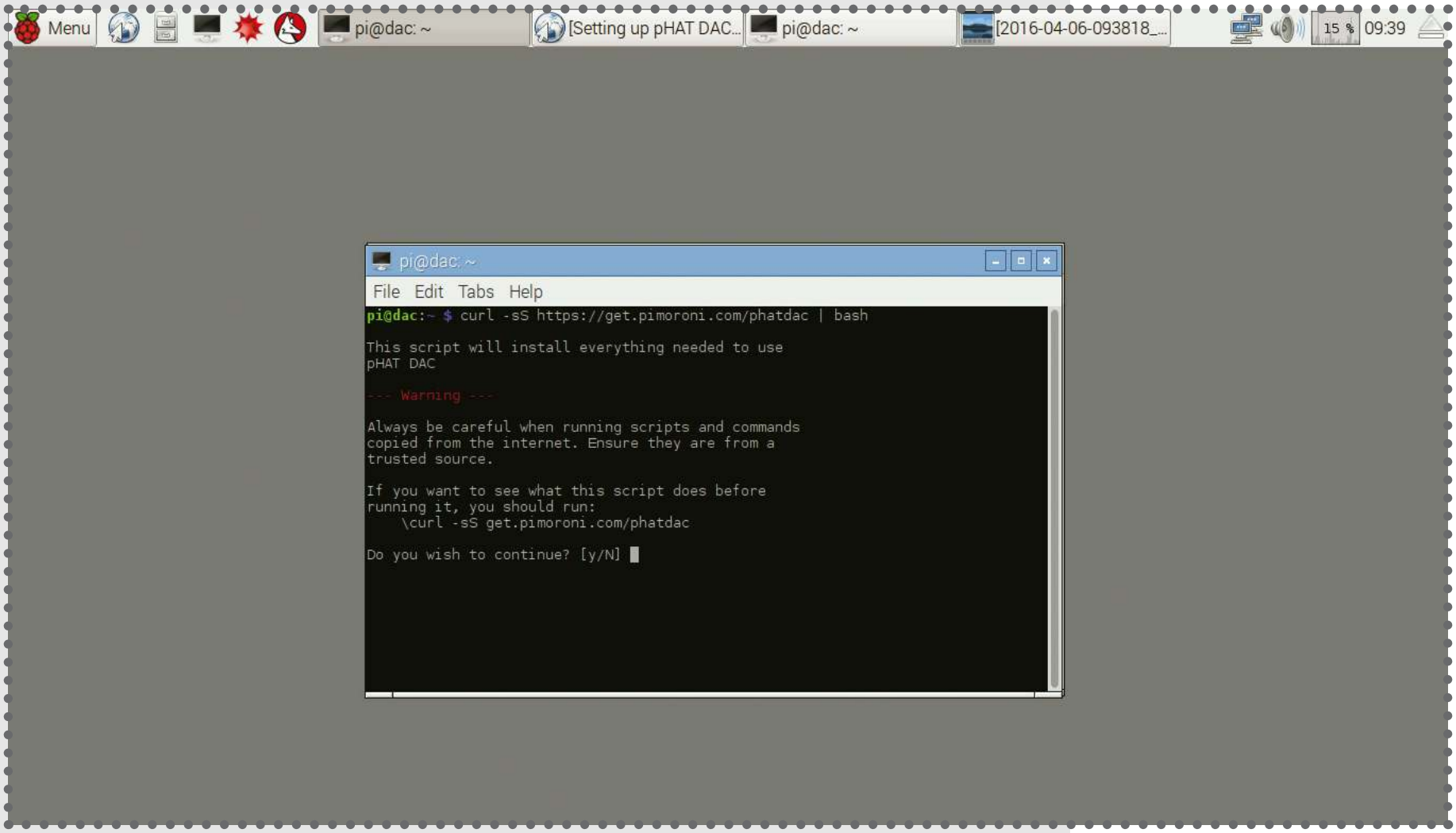
```
cd ~/pyPlaylist
./add_stations.sh
```

Auto-start on Raspbian

In Raspbian/Jessie the controversial systemd software was added giving a highly modular way of managing start-up scripts amongst other things. While systemd configuration files are now best practice, they can take time to fully understand. For that reason we would suggest using cron to start the script on reboot as a temporary measure.

crontab -e

@reboot /usr/bin/python /home/pi/pyPlaylist/app.py



Above Installing drivers is surprising simple

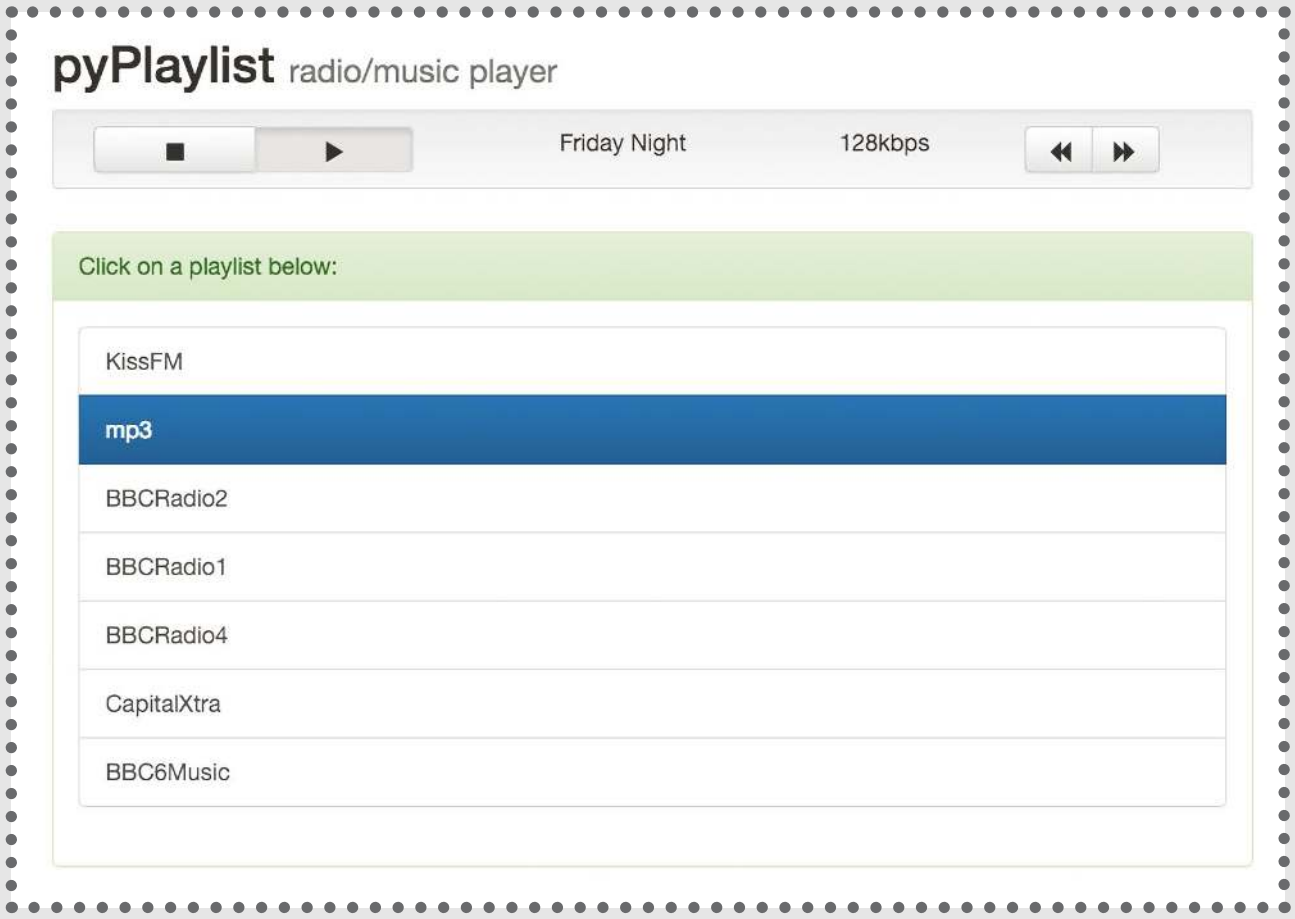
06 Get a new LIRC file

Each station is added into its own playlist – the mpc ls command shows you which playlists are available to play:

```
$ mpc ls
BBC6Music
BBCRadio1
BBCRadio2
BBCRadio4
CapitalXtra
KissFM
```

If you want to remove one of the stations then type in the following:

```
mpc rm KissFM
```



left pyPlaylist is a responsive and mobile-ready web server

07 Starting the web-server

Now that we have some stations, we can run the web-server from the pyPlaylist directory. Then open up a web browser to start playing a radio station. The following command reveals your IP address on Raspbian:

```
$ ./raspbian_get_ip.sh
192.168.0.20
```

Once you know the IP address, connect to the URL in a web-browser on port 5000, for example:

```
http://192.168.0.20:5000/
```

08 Add a custom music playlist

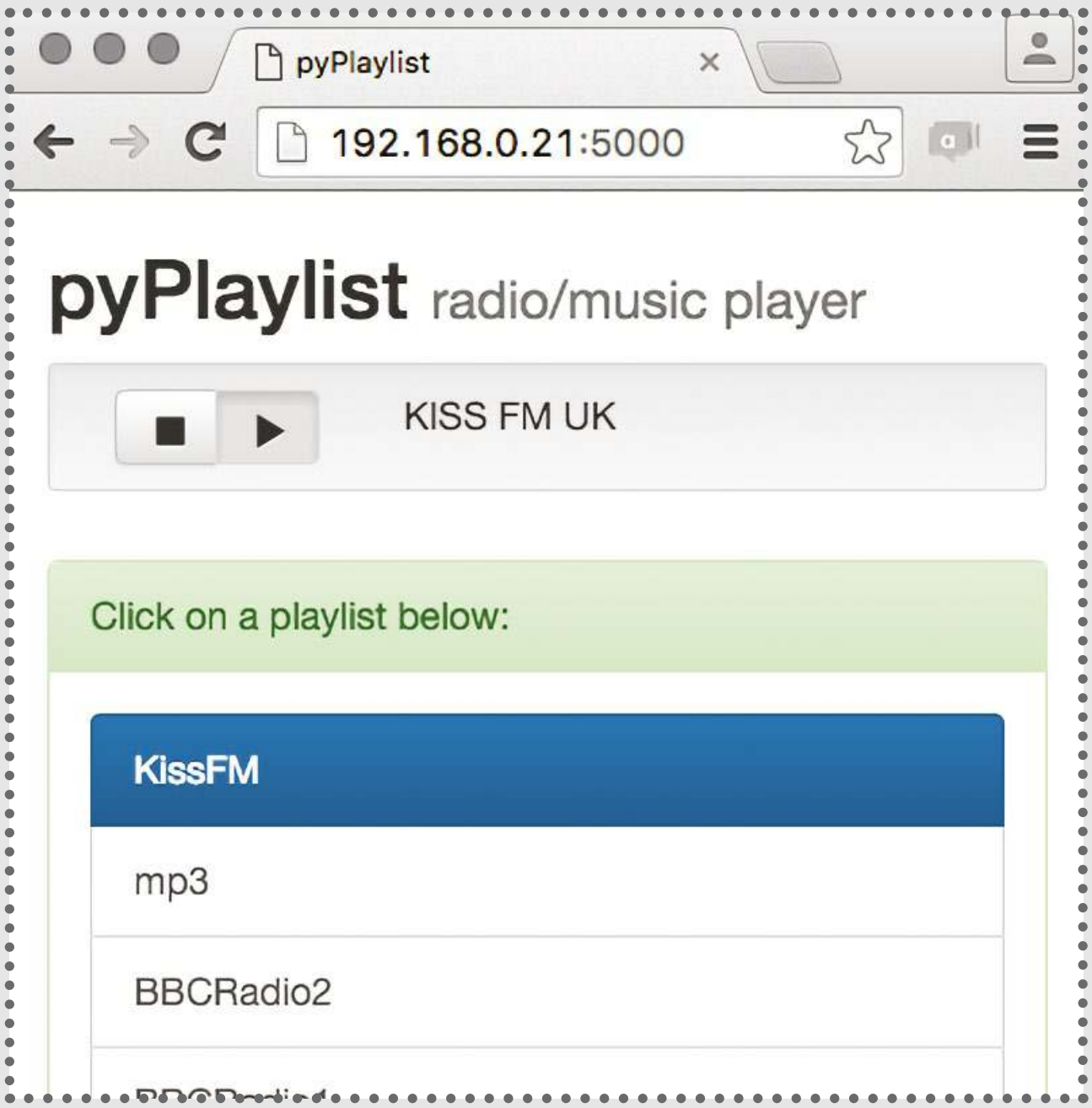
Now put together a sub-directory with your music files under /var/lib/mpd/music/ and ensure that mpd:audio has access to read it. Then we update mpd's database, clear out the current playlist and add in all the

tracks from the new directory (ambient) finally saving it as a brand new playlist.

```
mpc update
mpc clear
mpc ls ambient | mpc add
mpc save ambient
```

09 Finishing up

Now your music player is functioning, all that's left to do is to add some speakers, obviously! Almost anything with a RCA or 3.5mm input source will work just fine for this purpose. That part we will leave up to you. Now get ready to turn up the volume and enjoy the tunes!



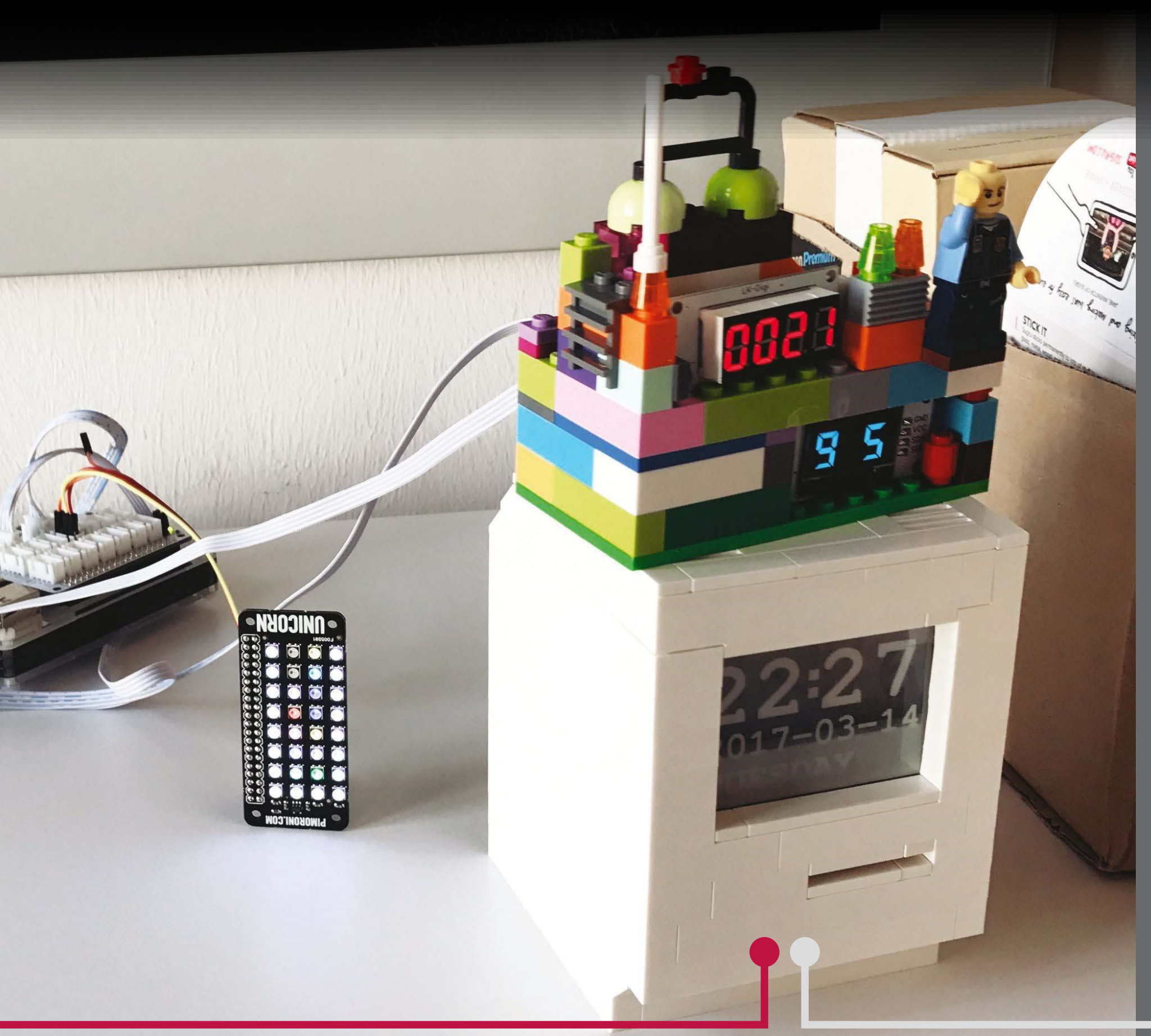
PyPlaylist

We wrote pyPlaylist with the Python flask framework which is an ideal starting-point for simple RESTful websites. The front-end code saves the screen from completely reloading by using jQuery to update the song or radio information. Bootstrap has been employed to make the pages responsive (compatible with your PC, phone and tablet). The code has been released under GPL, so why not fork the code and tweak it to your own needs?

left Once you're complete, it's easy to stream your favourite radio stations wirelessly to your hi-fi system

LEGO Macintosh Classic

This recreation of an old Mac in LEGO is a heady mix of contemporary technology and childhood nostalgia



What inspires your projects?
Pretty much all of my side projects are born from scratching an itch. In the past, that has been mostly software – starting with a blogging software called s9y in the early 2000s and more recently an iOS fitness tracking app called Gym Hero [...]. The LEGO Macintosh classic was my first hardware project, and it started because I live in Berlin and take the bicycle to work. I wanted to have a little display in the kitchen that tells me what the weather would be like. I purchased two small displays and a Raspberry Pi and made them display the carbon dioxide level in the living room and the current temperature on my patio. I then realised that this doesn't help me much in the morning when I have to decide whether or not to wear my elegant cycle-in-the-rain suit.

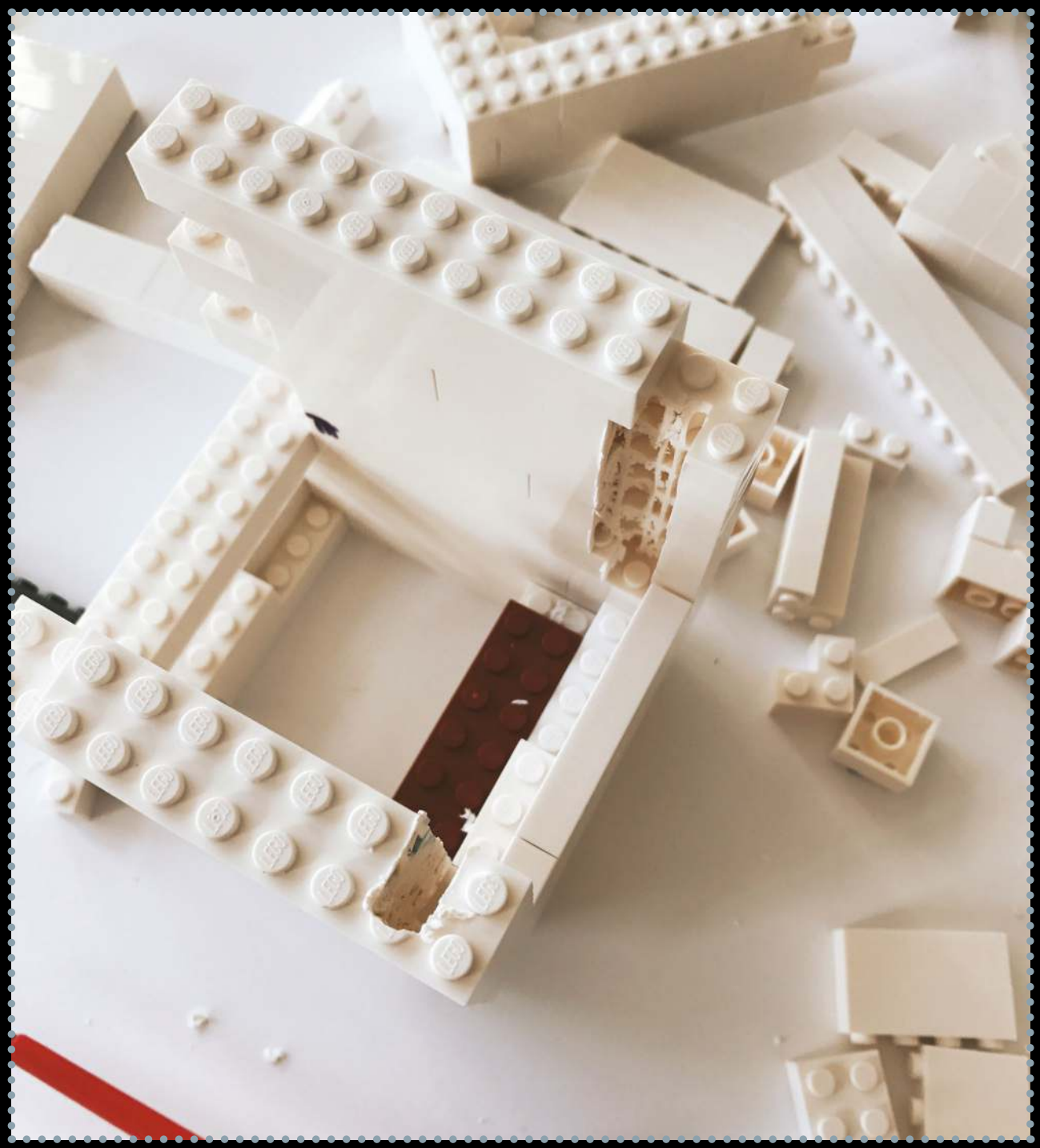
What was the thinking behind picking that particular e-paper display?
Since I needed more screen real estate to display the weather conditions for the day, I looked into what kind of displays are available for the Pi. Even though there are great, high-PPI colour displays, I opted for an e-paper display [...]. The weather condition kitchen display only needed to update every other minute while the lack of colour made it easier to design how the information is displayed. The great strength of e-paper displays is how unobtrusive they are; they just don't look like a computer and they don't glow in the dark. And that's just perfect [...]. I actually purchased two different sizes and used the smaller one for this project. The other is a 4.3-inch display from

Jannis Hermanns is a software engineer at moviepilot.com who traded his hobbies of kitesurfing and basketball for a bunch of kids and Raspberry Pis.

Waveshare and I've already begun planning to build another, slightly larger, LEGO Macintosh classic around it.

Can you tell us about the solution you came up with for updating the screen remotely?

There's a couple of tiny applications running on the Pi that take care of fetching the data, and one that updates the data that is displayed every minute. So the Pi is pretty much self-sufficient; there is no external entity updating its screen. It does all the work itself. The deployment of code is very easy and can be done remotely, [...] because the device is managed by resin.io's free tier.

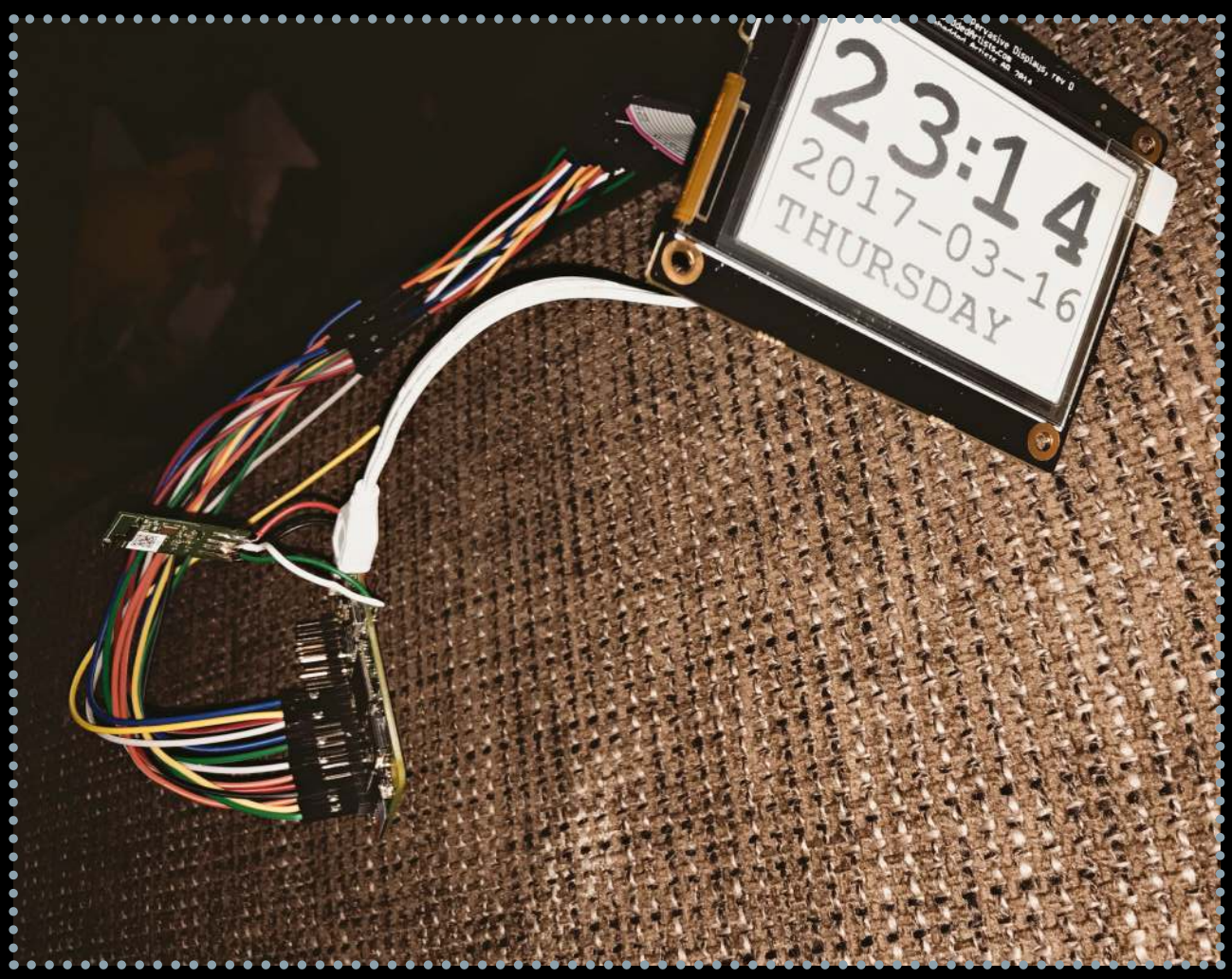


THE PROJECT ESSENTIALS

- LEGO Digital Designer
- LEGO bricks
- e-paper display by Embedded Artists
- Pi Zero (but suggest Pi Zero W now)
- Power supply
- Tiny retro Macintosh stickers
- Dremel
- Cables (male-to-female)

What was the most challenging part of the project?
That one's easy: to find the time. I worked on it every other night for two months, when the family was asleep (so they couldn't judge me for playing with LEGO in the kitchen). This usually involved working on the project for 30 minutes to two hours at a time. The rest was straightforward, but of course that's mostly because I'm a programmer. For someone who's new to programming, the whole software part of this project will take a little longer – but it's very much doable, don't be shy. And now that you know my name, you can [ask me to] help you.

If people want to have a go at your project, what tips would you give them?
Get a Raspberry Pi Zero W. It comes with Wi-Fi and Bluetooth on board, so you don't have to do any



Left The Raspberry Pi Zero W was yet to be launched when Jannis started his project, which is why there's a USB Wi-Fi dongle attached to the Pi Zero board in this photo. If you're trying this project now, he recommends using just the Pi Zero W

soldering there. Also, do not post images of severed LEGO bricks on the internet, especially if you're the one who Dremeled into them. As I've learned through this project, there are AFOL [adult fans of LEGO] on the internet who will belittle your abilities and judge you harshly if you cut bricks [...].

Which base image did you use in the end?
I used resin/raspberrypi-python:2.7 provided by the fine folks at resin.io. On my GitHub account you can find a docker image – http://bit.ly/epaperDockerfile – that is preconfigured for displaying data with the embedded-artists e-paper display that I used for this project.

What were you writing on your friend's screen? Did we spy your Gym Hero app being added to the Pi?
It currently displays the age of his daughters in days, the precipitation probability for the day, the current perceived temperature, as well as the min/max for the day. And since I made the Gym Hero app together with my buddy Jannik, who received the LEGO Mac as a birthday gift, it also displays how many workout plans we sold in the app the day before.

What has sparked your interest since finishing the Macintosh Classic?
It's a new device by my favourite Raspberry Pi shop: Pimoroni (https://shop.pimoroni.com). It's a 16×16 RGB pixel matrix called the Unicorn HAT HD that is ready to use – just plug it onto your Pi [...]. While you wait for your Unicorn HAT HD to arrive in the mail,

Like it?

Not surprisingly, the Venn diagram of nostalgia for Macs and contemporary makery creates an intersection that's packed with projects. One of Jannis's favourites is Måns Jonasson's NESMAC, which is an old Macintosh Classic with a Raspberry Pi inside(https://youtu.be/q1APtxbUfg8).

you can try it out [...] with a Unicorn HAT simulator I wrote (https://github.com/jayniz/unicorn-hat-sim) [...]. So far, I've built a little gifbox that allows you to drag a GIF onto it with your web browser. The next step is turning it into weatherbox that displays 16×16 animated pixel art showing you the weather for the day (sunny, cloudy, rainy, and so on). There it is again, the weather. I had no idea I'm obsessed with weather until this interview.

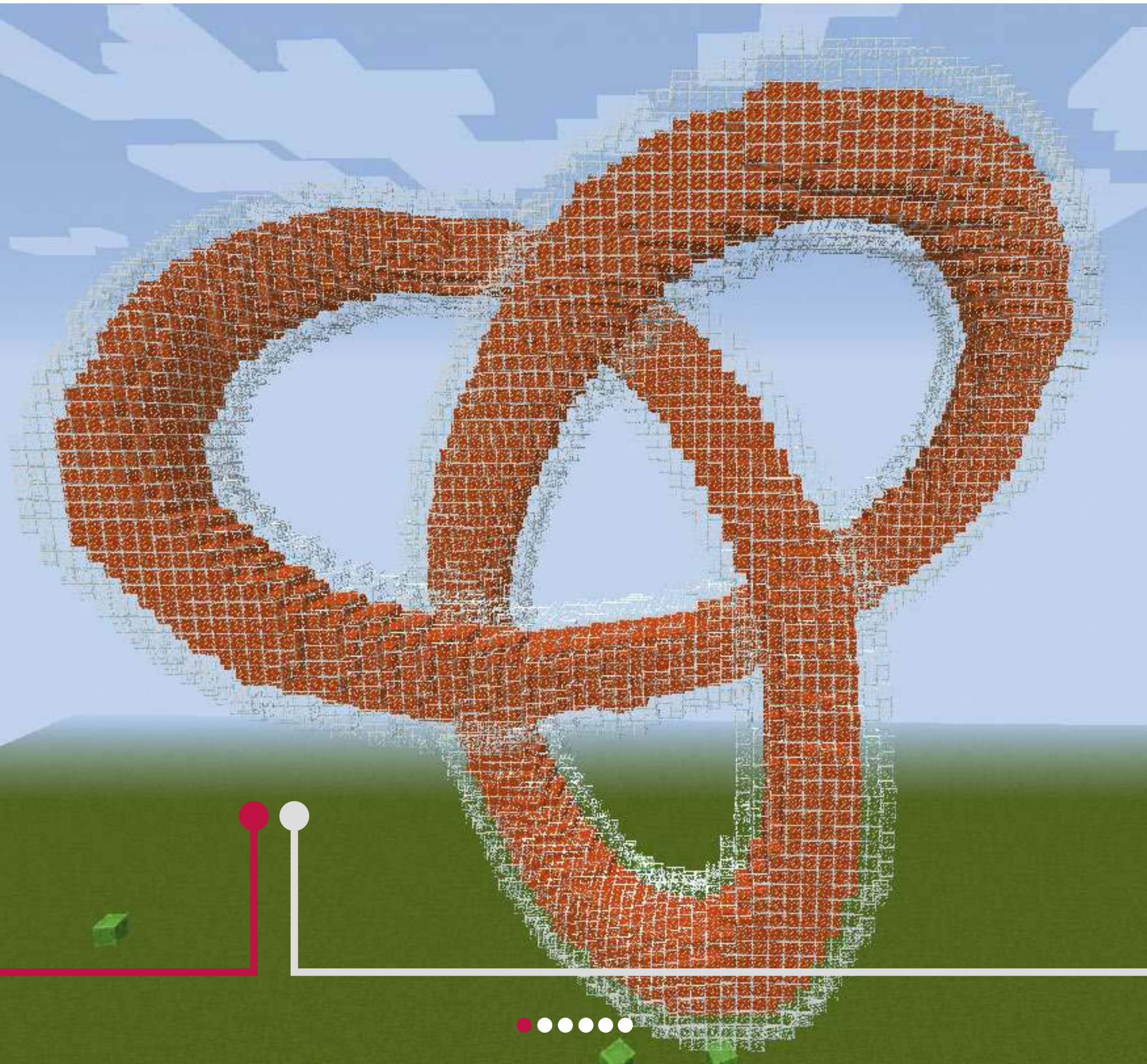
Further reading

To stay informed about the gifbox and weatherbox project mentioned in the interview, or any Jannis's projects, head to his website: https://jann.is. On his Ghost blog you'll find a complete write-up of the LEGO Macintosh Classic. You can also follow him on **Twitter: @jannis.**

Left The project wasn't all plain sailing. Jannis had to pull out a Dremel when he realised that he'd created a model that was exactly one brick too wide

Use Python to code new creations in Minecraft

Tap directly into Minecraft with Python and produce fantastic creations using Forge mod



Sometimes, Minecraft can seem far more than just a game. It's an incredibly creative tool and with the use of Redstone and Command Blocks you can produce some amazing worlds. We're taking things a step further by enabling you to plug Python code directly into Minecraft. What you do with it is completely up to your imagination! MCPiPy was developed by 'fleap' and 'bluepillRabbit' of **https://mcpipy.wordpress.com**, to connect MineCraft Pi Edition with Python on the Raspberry Pi, using open APIs.

However, with the use of Forge we have put together a package that enables the use of Python in retail Minecraft. We're using Raspberry Jam developed by Alexander Pruss, a Forge mod for Minecraft which implements most of the Raspberry Juice/Pi API.

THE PROJECT ESSENTIALS

Minecraft
http://www.mojang.com/games

Python
https://www.python.org

McPiFoMo
http://rogerthat.co.uk/McPiFoMo.rar

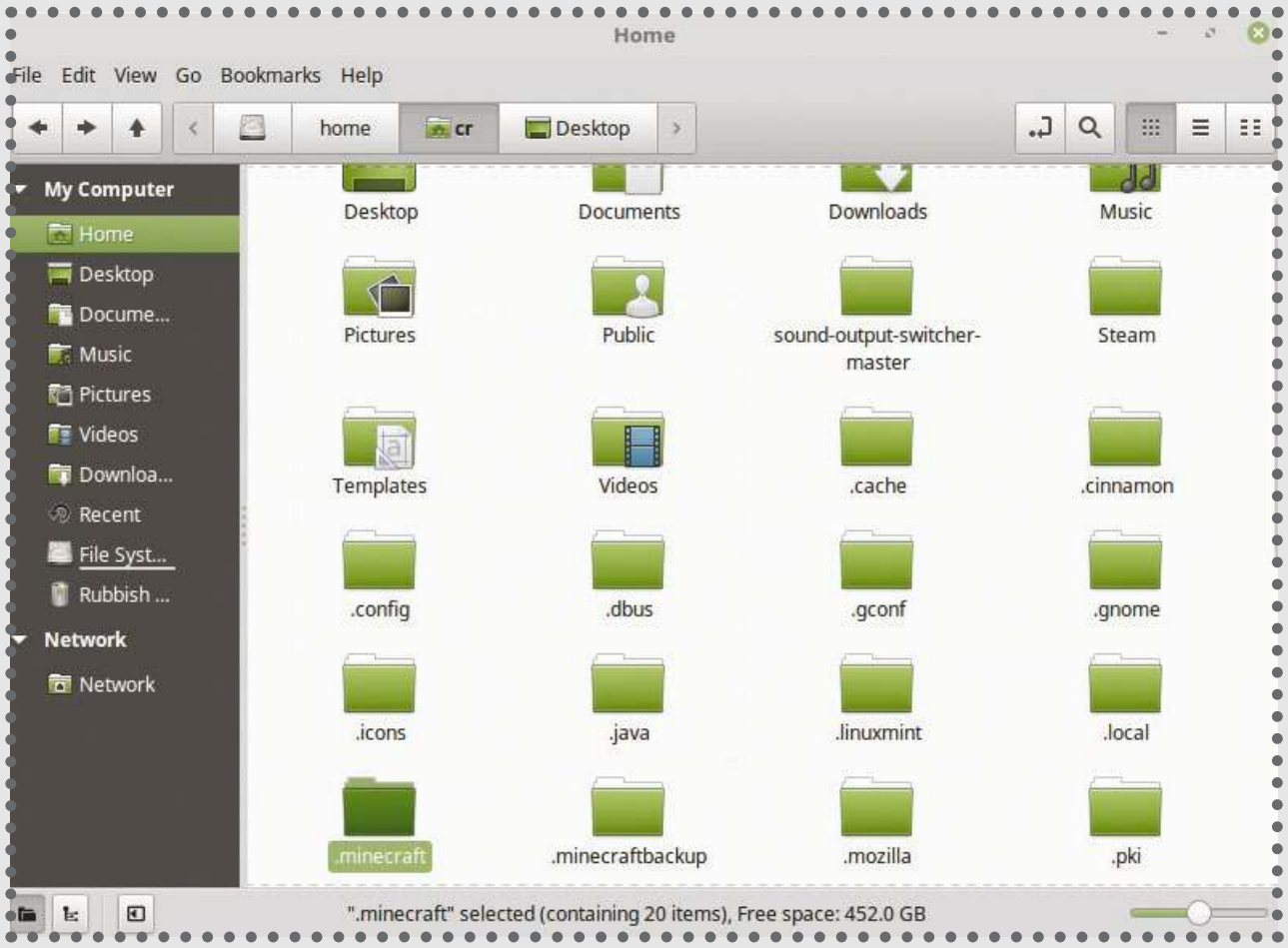
01 Replace your .minecraft directory

Backup .minecraft in your home directory. If you're using a GUI, you may need to press CTRL+H to view the hidden directories. In terminal mv ~/.minecraft ~/minecraft-backup should suffice.
Extract the new .minecraft directory from McPiFoMo directly into your home directory. If you have worlds you'd like to carry over, copy the Saves directory from your backup .minecraft directory into the new one with cp -r ~/.minecraft-backup/Saves ~/.minecraft/

02 Launch Minecraft in Forge mode

Launch Minecraft as you normally would, but after logging in, select the Forge profile. This should load Minecraft 1.8 with Forge 11.14. You can play around with the latest version of Minecraft and download and install an updated Forge if you wish, but these are the versions

we've found most compatible with Raspberry Jam. You'll know you're running the correct profile when you see the version numbers in the bottom left corner of the window. Create a new super flat world in singleplayer creative mode and you're ready to begin coding. We've included a single 'Flat' world pre-installed with the McPiFoMo package.



03 Hello World – chat commands

Using your favourite text editor, you'll need to create a new **helloworld.py file** and save it in **~/.minecraft/mcpipy** directory:

```
from mc import *
mc = Minecraft()
mc.postToChat("Hello world!")
```

Return to Minecraft and type **/python helloworld**

Minecraft will now run your python script, which should result in a chat command saying Hello world!

04 Create blocks

Now, by using setBlock() and getPos() commands we can place blocks into the world relative to our player's position. Try adding the following two lines to your helloworld script:

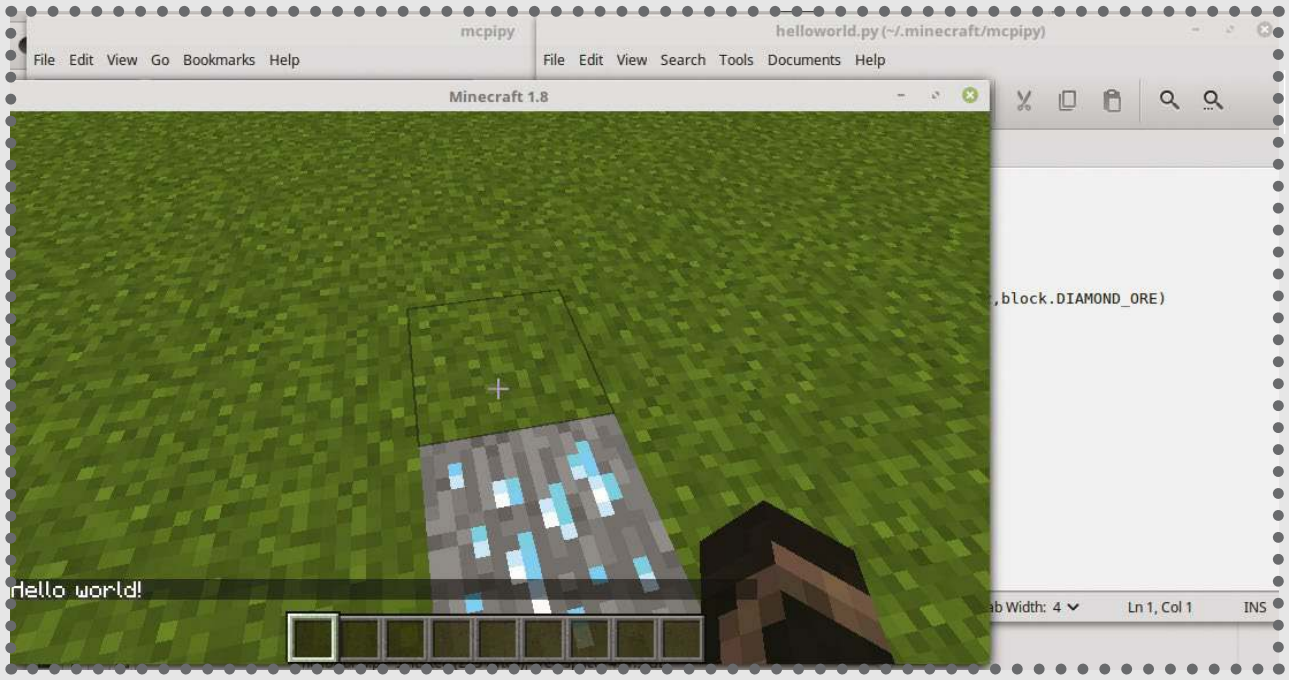
```
  playerPos = mc.player.getPos()
  mc.setBlock(playerPos.x,playerPos.y-
    1,playerPos.z,DIAMOND_ORE)
```

Then run **/python helloworld** again in Minecraft. You'll see the chat message again, but this time if you look down to the ground below your player character, you'll see a diamond block has also been placed at your feet. You can try replacing **DIAMOND_ORE** with any other Minecraft block ID (i.e. DIRT/GRASS).

05 Mmm, doughnuts

One of the pre-fab scripts that you will find in the MCPiPy collection is the doughnut:

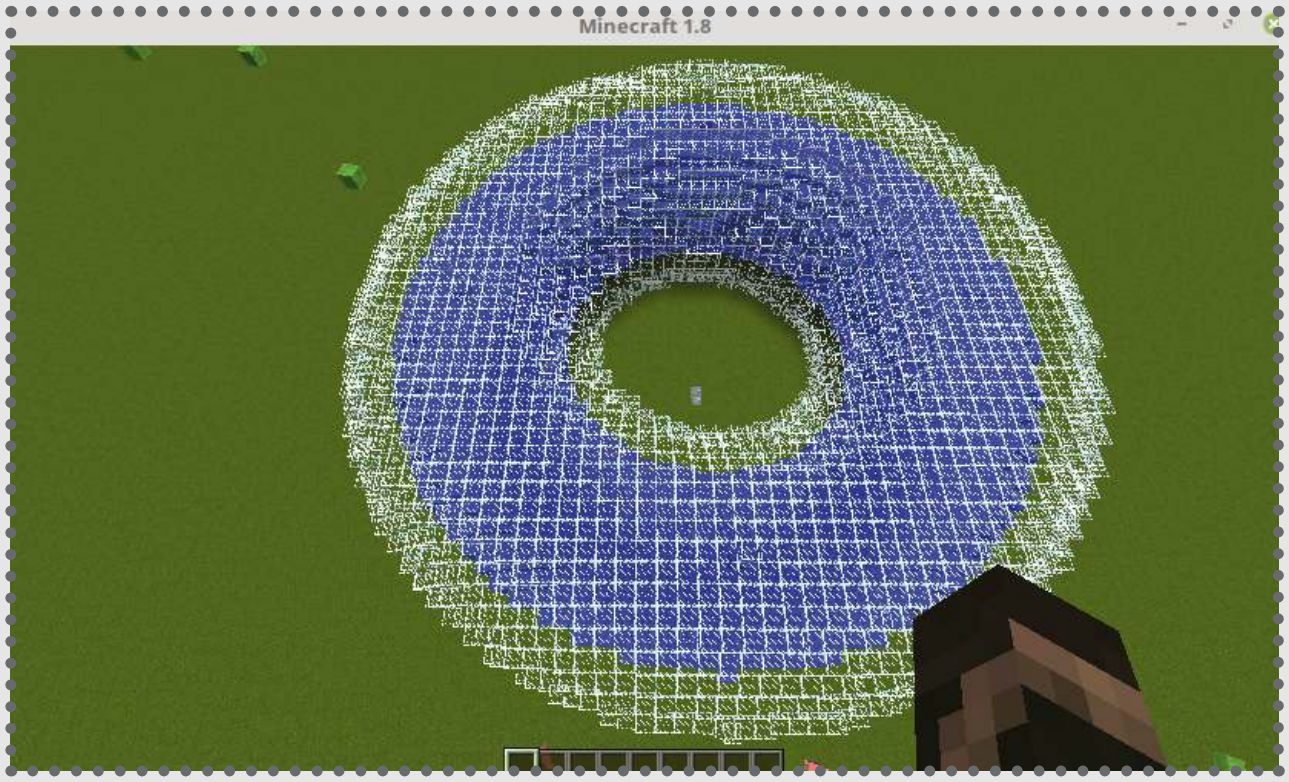
```
from mc import *

 def draw_donut(mcx,mcy,mcz,R,r,mcblock):
    for x in range(-R-r,R+r):
       for y in range(-R-r,R+r):
          xy_dist = sqrt(x**2 + y**2)
          if (xy_dist > 0):
```




```
    ringx = x / xy_dist * R #  nearest point
    on major ring
                ringy = y / xy_dist * R
                ring_dist_sq = (x-ringx)**2 +
    (y-ringy)**2

                for z in range(-R-r,R+r):
                    if (ring_dist_sq + z**2 <=
    r**2):
                        mc.setBlock(mcx+x, mcy+z,
    mcz+y, mcblock)

  mc = Minecraft()

  playerPos = mc.player.getPos()

  draw_donut(playerPos.x, playerPos.y + 9,
    playerPos.z, 18, 9, GLASS)
  mc.postToChat("Glass donut done")
  draw_donut(playerPos.x, playerPos.y + 9,
    playerPos.z, 18, 6, WATER_STATIONARY)
  mc.postToChat("Water donut done")
```

By changing the block ID from **WATER_STATIONARY** you can fill the doughnut with any object type. Try filling the glass with lava. Then try changing outer shell from glass to TNT.

06 Common errors



If you get a 'Script not found' error, this probably means that you don't have the mod scripts installed in your Minecraft directory. Check that you've replaced **.minecraft** with the one from McPiFoMo.

If you receive a 'Cannot run program "python"' error, your game cannot locate Python. Ensure you've got the latest version of Python installed, and that it's installed in Path. In the Bash shell type **export ATH="$PATH:/usr/local/bin/python"** to check. Should you come into any problems with memory leakage or infinite loops, you can stop a script that's running by just typing /**python.**

Cannot find script

If you see red text stating 'Cannot find script', check the name and location of your PY file. All of your Python scripts should be in ~/.minecraft/mcpipy, for Python and Minecraft to be able to locate them. You don't need to append '.py' to the end of your run command, just be sure you're using the exact name of your Python script file. /python doughnut will work just as well as /python doughnut.py, so long as doughnut.py is stored in ~/.minecraft/mcpipy. If you can't see this directory, remember to un-hide your files (CTRL+H).

Part 2: Command the seas with Sense HAT Battleships

The final part of our series on creating the classic game of Battleships using a Sense HAT on a Raspberry Pi

In the first part of this tutorial [see last issue] you started to build the classic game of Battleships, coding the features such as deploying ships, firing torpedoes and keeping a running total of your score.

In part two, you will begin by installing a small program which enables you to easily create animations to play across the Sense HAT's LED matrix. These can be used to introduce the game, display a game over message or update the player wherever your game requires it.

The tutorial then completes the main mechanics of the gameplay before setting up the code for deploying the game. Begin by introducing the game with a range of sounds and verbal announcements before scrolling a boat animation across the display. Then create a simple countdown from three to zero before the game begins. The game utilises the main function we created in part one. Once the game is over, we'll add the option to play it again or not. If you move the joystick down, the program plays a farewell sound, an animation is displayed and

THE PROJECT ESSENTIALS

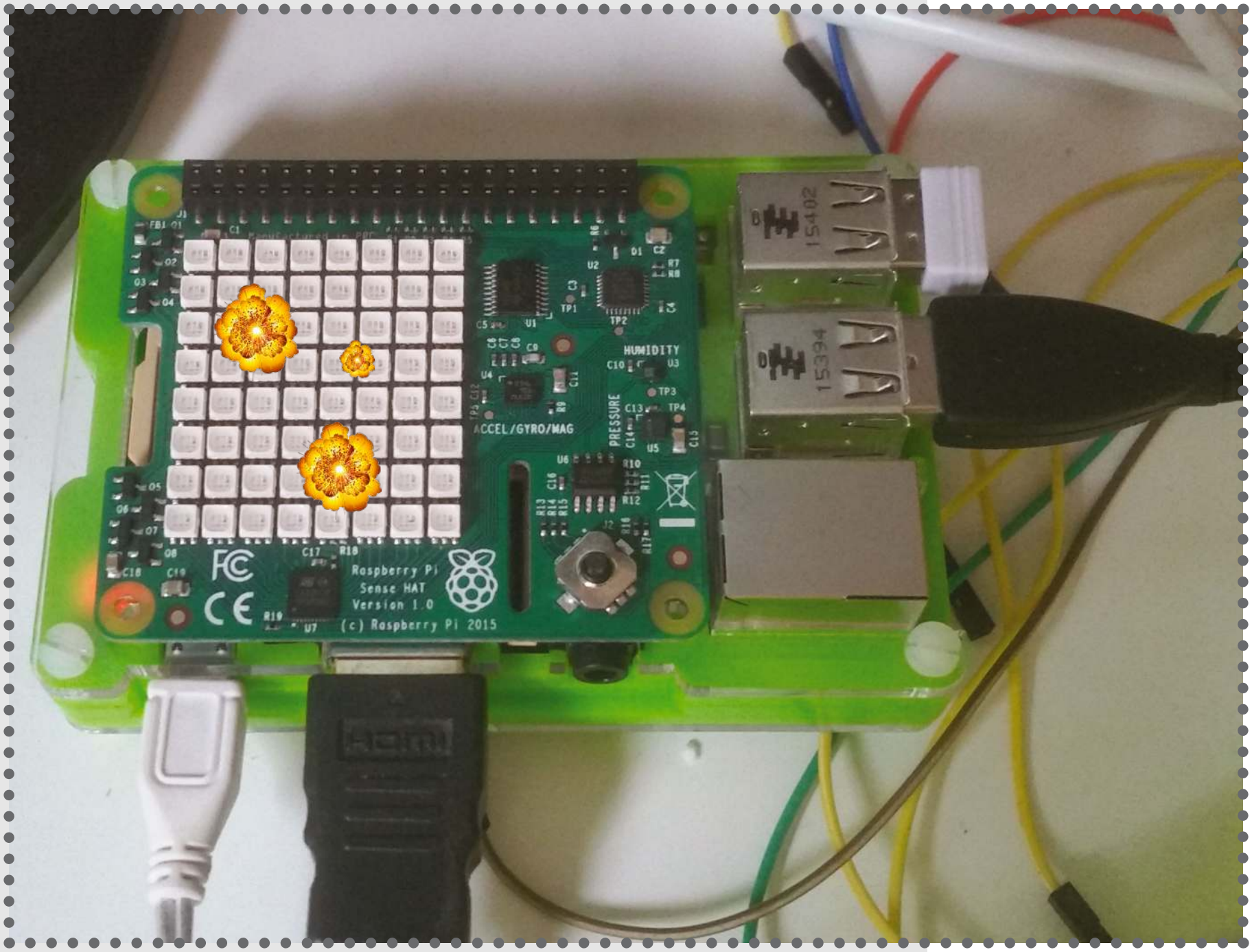
Raspberry Pi
www.raspberrypi.org/products

Sense HAT
http://bit.ly/BattleshipSenseHAT

then your game ends and the program closes. Move the joystick up and you can select to play another round and a new random layout is created. Check out the full game in action here: **https://youtu.be/SolWA1JHCac**.

01 Create your own 8×8 animation

There's a simple way to create an animation to run on the LED matrix: a superb onscreen program that enables you to manipulate the LEDs in real time. You can change the colours, rotate and export the animation as code. Begin by installing the Python PNG library, open the Terminal and type sudo pip3 install pypng. then:

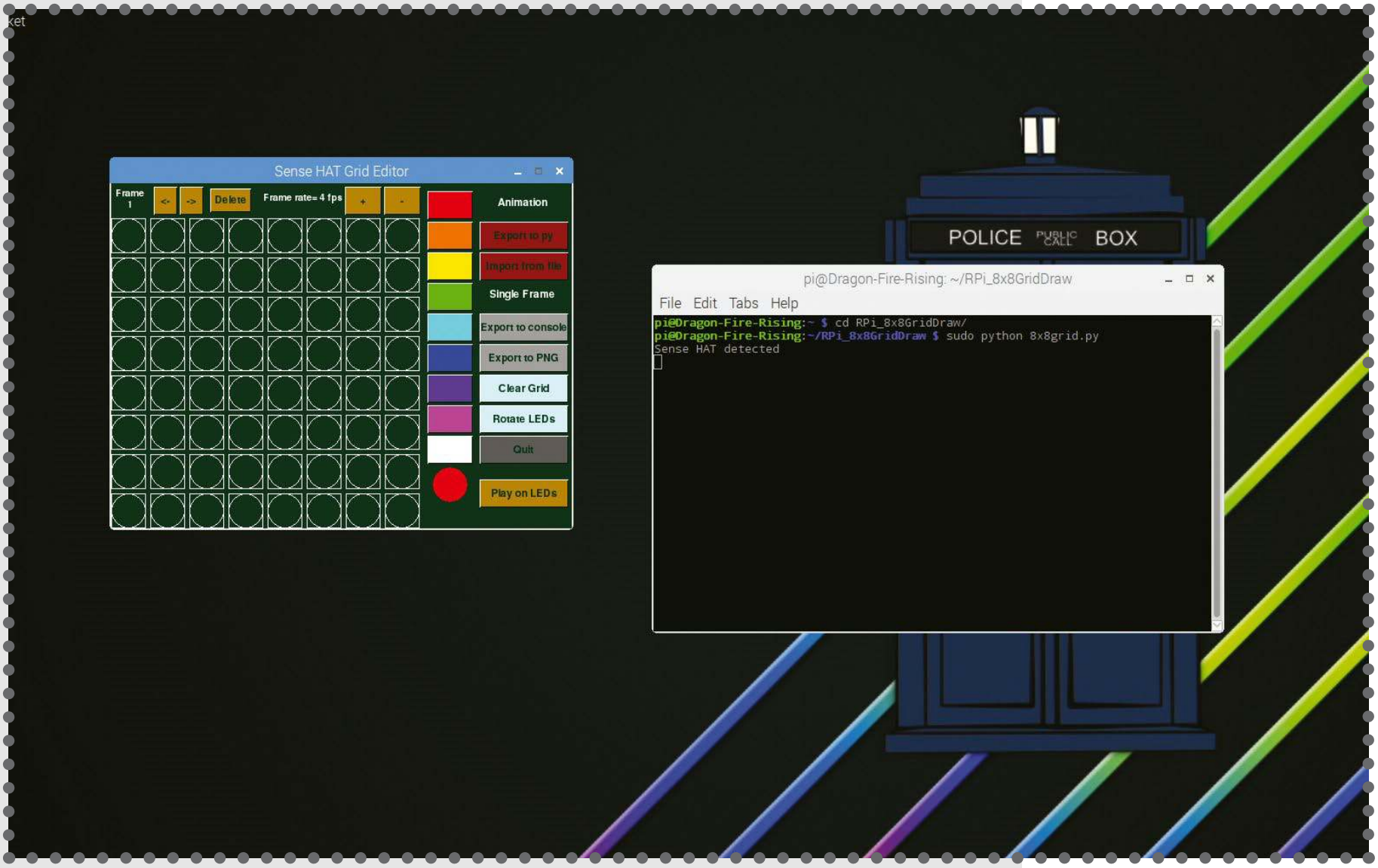


```
git clone https://github.com/jrobinson-uk/
RPi_8x8GridDraw
```

Once the installation has completed, move to the RPi folder, type ccd RPi_8x8GridDraw, then type python 8x8grid.py to run the application.

02 Create and export your animation

The Grid Editor enables you to select from a range of colours on the right. Choose a colour, then click the location of the LED on the grid. Select 'Play on LEDs' to display the 'pixel' on the Sense HAT's LED matrix. Clear the LEDs using the 'Clear Grid' button, then start over. Use the '+' button to add a new 'slide' and build up your animation. This can be a tricky process and so is well worth planning first. When you are finished, export the



animation to code pressing the 'Export to py' button. This saves the animation as a Python file in the **RPi** folder. It will be called **8x8animation.py**. You can create as many animations as you want; here we'll be making an introduction and the game-over screen.

03 Add the score

This tutorial now continues from the program you wrote in part one of the tutorial. Delete everything from below the 'Intro to the Game' comment – these lines were added to test your code. Now you can add the rest of the code. In line with the same level of indentation, add the code to play the 'score' MP3. This announces your score, line one. Use Pygame to play the MP3 file, line two; add a short pause to allow time for the sound file to play, line three.

```
pygame.mixer.music.load("sounds/yourscoreis.mp3")
pygame.mixer.music.play()
time.sleep(1)
```

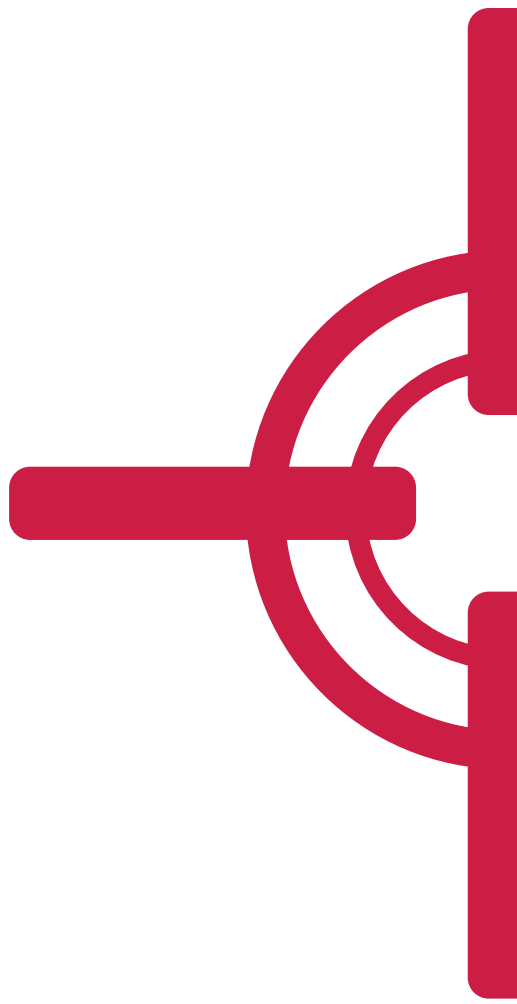
04 Display your score

Keeping the same level of indentation, add a line to scroll your score as a message across the LED matrix. However, first the score needs to be converted into a string in order to be displayed. You can edit the colour of the message by changing the three values between square brackets, up to a value of 255, line one. Then print the number of ships that are left to destroy, line two.

```
sense.show_message(str(score), text_colour=[0, 200, 255])
print ("There are", number_of_ships, "ships left")
```

Pixel Inspiration

Johan Viet has some excellent and inspirational examples of 8x8 pixel art (**http://bit.ly/8x8pixelart**), which include some famous characters and will show you what you can create with 64 pixels of colour.



05 How well have you done? Part 1

Now the program checks how well you performed during the game. If the number of remaining ships is zero, then you have done very well, line one. On line three, a suitable 'well done' message is scrolled across the LED matrix. Add a suitable 'winners' sound such as a fanfare, line four, and then play it. Remember to add a small delay for the file to play.

```
  if number_of_ships == 0:
    print ("WELL DONE, TOP JOB") #ADD SOME
    ANIMATION
    sense.show_message("TOP JOB", text_
    colour=[0, 255, 255])
    pygame.mixer.music.load("sounds/winner.mp3")
    ## add a sea sound
    pygame.mixer.music.play()
    time.sleep(3)
```

06 How well have you done? Part 2

How well have you done? Part 2

If you didn't sink all the ships then you didn't win the game and lost. You could add an extra conditional to check for a different amount, for example half the ships. Begin by scrolling a message across the LED matrix using the code on line two. Then play a suitable loser sound, lines three and four.

```
    else:
    print ("better luck next time")
    sense.show_message("BETTER LUCK NEXT TIME",
    text_colour=[0, 255, 255])
    pygame.mixer.music.load("sounds/nexttime.
```

```
##########################################################
########### Intro to Game  ###############################
##########################################################


### intro music ###
pygame.mixer.music.load("sounds/intro.mp3") ## add a sea sound
pygame.mixer.music.play()

import intro_ship
```

```
mp3") ## add a sea sound
pygame.mixer.music.play()
```

07 Starting the game

This completes all the game mechanics, which are stored in the function called main() – this code will only run when main() is called. However, before that, let's create an introduction to the game. Begin by playing a suitable sound, line one. This example is the soothing sound of waves. Trigger the sound to play, line two. On line three, use the code import intro_ship to import the animation you created in Step 2. This is displayed as the sound is playing, creating a nice introduction to the game.

```
pygame.mixer.music.load("sounds/intro.mp3") ##
 add a sea sound
pygame.mixer.music.play()
import intro_ship
```

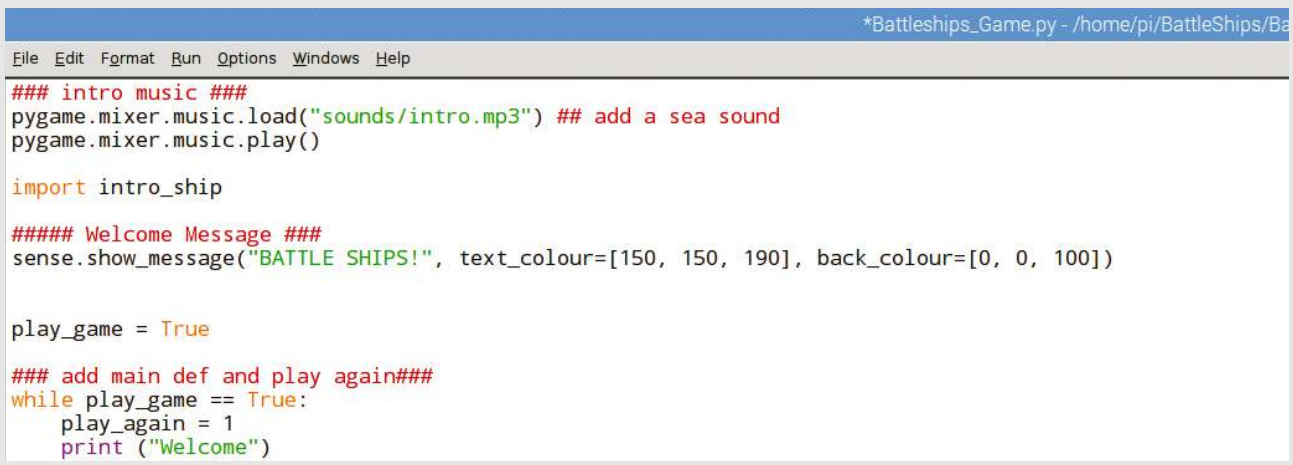
08 Beginning the game

After the short animation finishes, we announce the start of the game to the player. This uses the familiar sense. show_message code to scroll a message across the LED matrix. This time, add a background colour using the code back_colour=[0, 0, 100], line one. Create a variable, line two, which sets the state of the game in play as True, meaning the game is in play. Next, check that the game

is in play, line three, and create another variable called play_again, line four, to respond if a player wishes to play the game again.

09 Create the sea

Now to create the sea. This is a list containing 64 items which turn each matrix LED blue. Every time you select a position, the data from the Sea2 list, (created in tutorial one), an enemy ship or ammo dump is added to the current position in the Sea list. Then the LED display is updated. Begin by setting the colour and assigning it to a variable called S. Then create the list and add 64 'S' characters – note that a comma is not required after the last entry. To write the list data to the LEDs, use the code sense.set_pixels(Sea). Add a small delay and then stop the intro music from Step 7 playing. Note that all these



lines are indented in the code.

10 Get ready and countdown

After the introduction to the game, inform the player to 'get ready' as the game is about to begin. This is triggered by a simple sound file and uses the Pygame play code which you've used before. Next, create a simple countdown from three to one. This uses a loop to iterate over a list in reverse order. Each time, it displays

the number: sense.show_letter(str(num)). Note that the number is converted into a string before being displayed. As before, add a small delay between each number being shown.

11 Go

After the countdown, the game is ready to play. Inform the player with a simple 'go' or similar auditory command, lines one and two. Then call the main() game function that you began in tutorial one and completed in Step 6. Since the main() function is at the beginning of the Python program, it is already loaded and ready to deploy. The game begins…

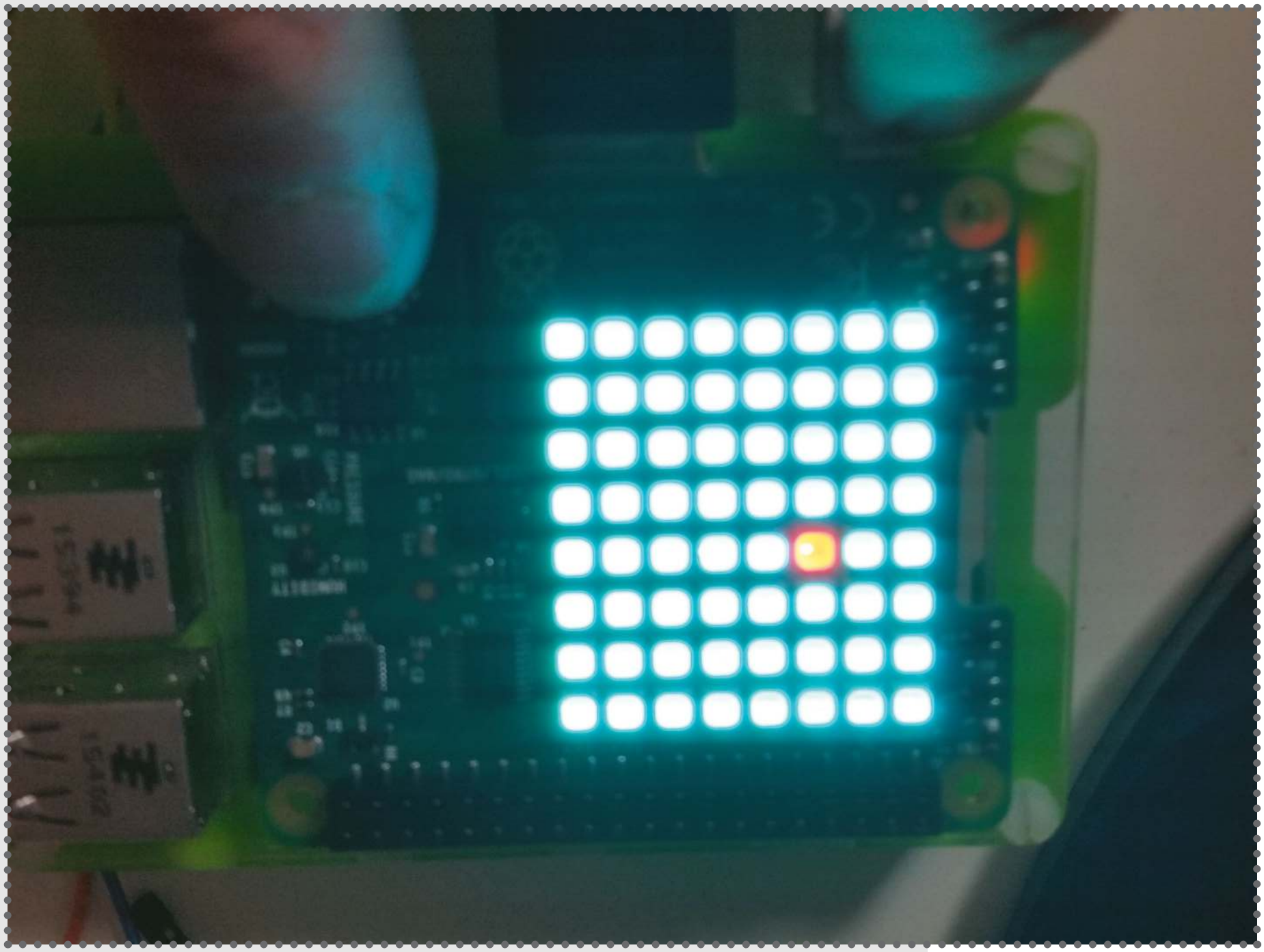
```
pygame.mixer.music.load("sounds/go.mp3")
pygame.mixer.music.play()
### Begin the main game ###
main()
```

12 Play again?

The game will now begin and you can select your chosen location in the sea. This position is checked to see if there is an enemy ship, ammo or just open space. Once the main() function has finished, it breaks out of the game loop and runs the code below. This gives you the option to play again. Begin by loading the 'choice' image. Then load and play a sound file asking the player if they want to play again.

13 Checking for player response

To respond to the question the Sense HAT joystick can be used to select up or down. On line one, create a



while loop which checks if the game choice is equal to a value of one or true. If it is then use Pygame to check for a downwards joystick movement. If you moved the stick down then you do not wish to play again; the play_again variable is therefore changed to a value of 0. The variable, play game from Step 8 becomes false to indicate that the game has ended.

14 End of the game

If the player chose to end the game then a suitable sound file is loaded and played, lines one and two. If you have created an ending animation, you can play this by importing the file, line four. Finally, break the loop so that

the program ends. To play the game again, you will now need to reload the program. Note that the indentation at this level is the same as the last code of the previous step.

```
    pygame.mixer.music.load("sounds/soon.mp3")
    pygame.mixer.music.play()
    print ("Bye Bye")
    import end_animation
    break
```

15 Playing again

The other option is to play the game again. This is achieved by moving the joystick up. Line one checks for this movement and then changes the play_again value to zero. Note that the play_game variable from Step 8, which indicates if the game is in play, is not altered and therefore stays as True. The program repeats the loop, checks the variable, finds it to be True and deploys the main() game function again.

16 Running battleships

This completes the tutorial and program running behind the Battleships game. Ensure that the sound files are stored in a folder named sounds. Also, ensure that the animations are stored in the same folder. If you organise your folders differently, you'll need to edit the code accordingly. Save your Python program and execute it by pressing F5. How many ships can you sink?

Creating images

The Grid Editor program used in Step 2 can also create static images. These can be displayed on the Sense HAT's LEDs. Follow the instructions from Step 2 to load up the Grid Editor tool and use it to place coloured pixels in the grid. Select 'Play on LEDs' to display the colour on the LED matrix. As before, clear the LEDs using the 'Clear Grid' button and then start over. Then, when you have finished, save and export it as a PNG file. Use the code sense.load_image("name_of_your_file.png") within your program to display the image.

The art of infrared photos

Discover the stunning but bizarre world of photographing the invisible with the Raspberry Pi and NoIR camera

Although we can't see it, infrared is all around us. If our eyes were sensitive to it, therefore, we'd be able to see the world in a different way and we can speculate that we might see some things in totally unknown colours. Even though it's invisible to our eyes, glass lenses are able to focus infrared (IR) and the CCDs in cameras are sensitive to it, so digital cameras have all that's needed to take IR photographs. As we're about to see, things aren't quite that simple with an ordinary camera, but the Raspberry Pi's NoIR provides an ideal solution.

Before delving into IR photography, though, it's worth considering why we might you want to shoot in IR. First of all, conventional IR photography is black and white, which has its own unique appeal. But while black and white photography normally contains a broad spectrum of greys, part of the attraction of IR is that it has a much higher contrast, which makes for very dramatic shots. There are two reasons for this. First, grass and the foliage on trees appear brilliantly white, almost to the point that they seem to be glowing. And second, blue skies appear virtually black, so a partially cloudy sky looks especially spectacular. As a result, IR photographs have a look which has been variously described as ghostly, other-worldly, ethereal or just plain spooky.

The snag with using ordinary cameras to photograph in the infrared is that manufacturers do their best to prevent their cameras from responding to IR. Because the presence of IR would cause colours to look unnatural, an IR blocking filter is placed over the CCD so that the camera only records visible light. However, that filter isn't 100 per cent efficient so, to differing degrees,

THE PROJECT ESSENTIALS

Raspberry Pi
Any model with a CSI camera port

Pi NoIR v2
Camera Module
http://bit.ly/PiNoIR

Camera adaptor
if using Pi Zero
http://bit.ly/ZeroAdaptor

Adafruit PiTFT 2.8-inch Plus TFT + Resistive Touchscreen
Choose correct version for your Pi model

http://bit.ly/PiTFTPlus

5V USB power pack

R72 infrared filter or equivalent

Case & mounting

most ordinary cameras are able to record IR, albeit with much less sensitivity than they have to visible light.

The challenge of infrared

To take a pure IR photograph, it's necessary to exclude all visible light which would, otherwise, interfere with the IR. This is done using what is often referred to as an IR filter, but is more accurately called an IR pass filter to differentiate it from an IR blocking filter. However, the main problem is that with most of the infrared cut out by the IR blocking filter, and with most of the visible light cut out by the IR pass filter, very little radiation of any type reaches the CCD. The bottom line is that the camera has to be set to a very high ISO speed and this results in a noisy image, and even then the exposure time is likely to be several seconds, even on a bright sunny day.

Despite the difficulties of using ordinary cameras for infrared photography, an ideal and low-cost solution is available to Raspberry Pi users in the form of the NoIR camera. NoIR means no IR filter, i.e. it doesn't have the IR blocking filter, and while it's mostly used in conjunction with IR LEDs for night-time security applications, it's also ideally suited to daylight infrared photography.

Having briefly mentioned that all-important IR pass filter, let's discuss the options. An IR pass filter allows infrared to pass while blocking visible light, but in practice things aren't quite that simple as filters differ in their cut-off wavelength. The least aggressive IR filter is called the R72 and it cuts off at 720nm, which means that a small amount of the deepest visible red light will get through in addition to the infrared. The result is that the effect isn't quite as dramatic as with some of the

filters with a longer wavelength cut-off, but this filter is widely available and cheap so it's where most people start. For many people the R72 is perfectly adequate, but if you strive for even more dramatic effects you could try an 850nm filter; the part number varies from one manufacturer to another, but will probably be something like R85 or IR 850.

Build and use an IR camera

Commonly, the NoIR v2 camera, just like its visible counterpart the Raspberry Pi Camera Module v2, is used in fixed installations, but for this application you need a portable unit. To achieve this you must assemble a Raspberry Pi, the NoIR and its IR filter, a small LCD display and a battery power source, and house them into a portable case. It also needs some software so that shots can be taken by touching the screen. Our step-by-step instructions outline one possible way you can achieve this, but there are lots of other options and, if you're an accomplished Pi hacker and relish a challenge, you might decide on a different approach.

Once you've assembled your kit, it's time to go into the great outdoors and try your hand at infrared photography. While you'll certainly want to experiment, a bit of guidance will help you get up to speed as soon as possible. First of all, the most unique aspect of IR photography is the ethereal glow of greenery, so it's best to start in spring or summer when the trees are in leaf. Similarly, because that characteristic look is much more pronounced with good illumination, choose a bright day, ideally with direct sunlight. The other contributor to the high contrast look of IR photographs is the almost black sky, but this isn't achieved on an overcast day so look out

for at least a partially blue sky.

The results, straight out of the camera, are often not as impressive as they could be, so a degree of digital image manipulation is needed to get the most out of your photos and turn them into true works of art. We're not going to provide step-by-step instructions because that would apply to one particular photo editor, but everything you need to do is pretty basic so you should have no difficulty in adapting our generic instructions to your package of choice.

Although taking shots with the NoIR through an IR filter will produce an essentially monochrome photograph, you need to get rid of any colour cast, so your first job is to convert the colour photograph to a genuine black and white (i.e. greyscale) image. Next, because IR photos often look quite lacklustre initially, you probably need to increase the brightness and the contrast, but do experiment to find the look that's right for you.

01 Connect the PiTFT 2.8-inch Plus

Connect the PiTFT screen to your Raspberry Pi. This couldn't be simpler but, to get it to work, you need to install a new Linux kernel with the necessary support. This can be downloaded from the Adafruit website, although the download in Step 2 includes this support.

02 Connect the NoIR

Next, you need to connect the NoIR camera to the Pi. Again this is a very simple process although, without some software support, you won't be able to preview

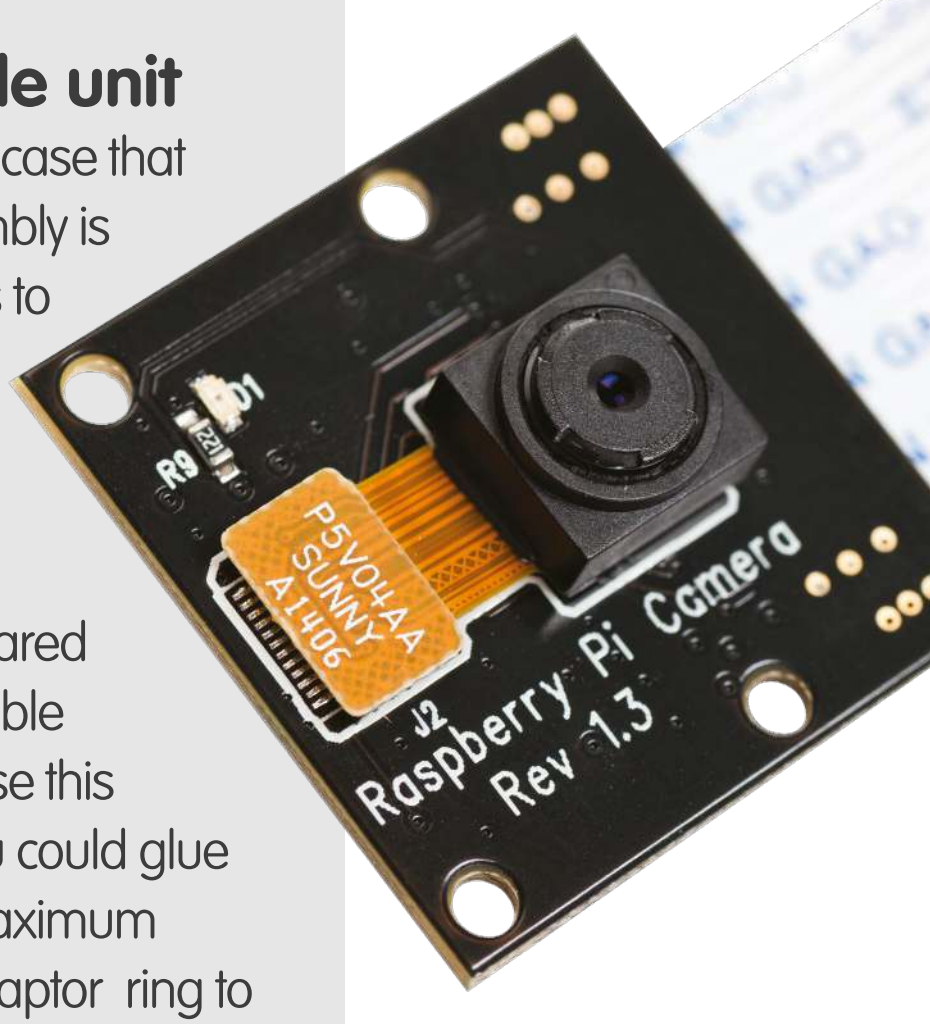
images on the PiTFT. Adafruit has published a project with an associated download which provides support for the PiTFT and also includes software to implement a point-and-shoot camera. This project – which you can find at http://bit.ly/DIYPicam – is for a visible light camera but, since the NoIR is identical to the standard Pi camera except for its lack of an IR blocking filter, the software is just fine for this project.

03 Add a power supply

You'll need a 5V battery pack and, while USB power supplies are widely available, you'd only be able to turn your camera assembly on and off by plugging in and unplugging the power pack from the Raspberry Pi. This is inconvenient, so you'll need to wire a switch into the relevant wire in the lead between the power pack and the Pi.

04 House everything as a portable unit

The trickiest part of this project is to provide a case that holds all the parts together so that the assembly is easy to handle and robust. The best option is to obtain a general-purpose instrument case of adequate dimensions and adapt it by providing fixings for the various components and making holes for the TFT screen and the camera lens. You also need to attach the infrared filter in front of the lens with no leakage of visible light round the edge. If you're only going to use this configuration with a single type of IR filter, you could glue the filter over the lens hole in the case. For maximum flexibility, glue an appropriately sized filter adaptor ring to the case so you can screw filters on and off.

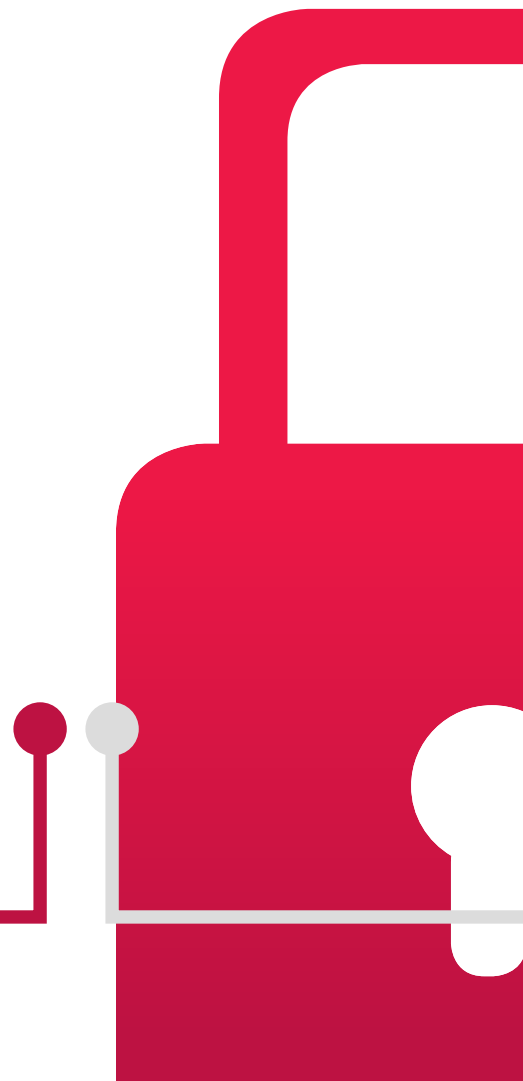


Monitor your car with a Raspberry Pi

With just a few extra items, you can use your Raspberry Pi to build a very sophisticated monitor for your automobile

Many people have developed projects where Raspberry Pis are used as the core of house-monitoring systems. There is no reason that you can't go mobile with this concept and create a monitoring system for your car. This month, we'll look at how you can build such a monitor and keep track of what your car and its engine is doing. The reason this is possible is because of the onboard diagnostics (OBD-II) protocol. Essentially, every modern vehicle has an OBD port so you can connect a device and interact with the control computers within the engine. In most cases, you'll need a dongle to act as an interface between the vehicle and some type of computer. The dongles can usually be connected to through a USB cable or Bluetooth. For this project, you could use either. If you do choose to use the Bluetooth dongles, however, don't forget to install the Bluetooth stack for your Raspberry Pi. In Raspbian, you can do so with:

```
sudo apt-get install bluetooth bluez bluez-
    utils blueman
```



You'll also need to have a Bluetooth dongle plugged into one of the USB ports on your Raspberry Pi, unless you have a Pi 3 with built-in Bluetooth. You'll also want some kind of display to show what is happening inside your vehicle's engine. You can either choose one of the purpose-built screens, which plug into the display connector on the Raspberry Pi board. Or, you can select another display type that can connect over HDMI. You will also have to come up with some mechanism to mount this display within the confines of your particular vehicle.

Once you have the physical elements connected together, it's time to start getting your code talking to your vehicle so that you have data to display. The first step is to install the OBD Python module. It currently isn't in the package list for Raspbian, so you will need the following command to install it using pip.

```
sudo pip install obd
```

This module is designed to work with the standard ELM327 family of OBD adaptors. Also, the OBD module is sub-1.0.0 in version number, so you should keep an eye on the changelogs for updates to see if any changes will affect the features you are using. You can find the full documentation at http://bit.ly/PythonOBD. Communications with the OBD interface happens with a core object, named OBD. Interacting with your vehicle's engine involves a query and response system. A 'Hello World' type of program would look like the following:

```
import obd
my_car = obd.OBD()
cmd = obd.commands.SPEED
response = my_car.query(cmd)
print(response.value)
```

The call to OBD() instantiates a new connection object to your vehicle over USB or Bluetooth. You can then select one of the query commands included with the OBD Python module. You send this command to your vehicle with the query() method and get the response back as an object. You can then read off various parameters, such as the value . There are also helper methods within the response object. For example, this code prints the current speed in miles per hour.

```
print(response.value.to("mph"))
```

The initialisation scans all devices connected to your Raspi and tries to bind to the first one it finds that appears to be an OBD interface. But, you could have multiple devices connected, or you may have a less standard interface. In these cases, you can control the connection more explicitly: set the port, the baud rate and the protocol for your connection. For example, you could connect to an OBD dongle on USB with the command:

```
conn = obd.OBD('/dev/ttyUSB0', 9600)
```

If you have multiple OBD devices connected to your Raspberry Pi, you can get a list of them, and then connect to a given one, with the following code.

Why Python?

It's the official language of the Raspberry Pi. Read the docs at python.org/doc

```
obd_devices = obd.scan_serial()
print(obd_devices)
curr_conn = obd.OBD(obd_devices[0])
```

If you need to verify the connection to your vehicle before interacting with it, you can use the method is_connected() to get a true or false for the connection. There is a rather long list of possible commands that you can use in the query() method. They are available in the table 'commands', as part of the obd Python module. If you already know the name, you can either use it as a property of the 'commands' table, or you can explicitly refer to it by using list syntax. For example, you could refer to the RPM value of your engine with either of the following lines of code.

```
rpm1 = obd.commands.RPM
rpm2 = obd.commands['RPM']
```

The query() method returns an OBDResponse object with the details gathered from your vehicle. These response objects have four properties. The value property contains a decoded value from the vehicle's response. The command property contains the query command that was initially sent to the vehicle. The message property contains a Message object, which holds the raw response from the vehicle's system. The last property is named time, which contains a time-stamp of when the response was received. One thing to be aware of is that if the query fails to get a response from the vehicle, the obd module will return

an empty response object. You can check for this by using the response object's is_null() method. For most cases, the returned value in the response object is a simple value. However, there is a special query command that returns a much more complicated response; namely, the status query command. In this case, you get a response object whose value contains a large amount of information that you can use right away. For example, you can find out whether the check engine light is lit with the following code.

```
if response.value.MIL:
  print("Houston, we have a problem.")
```

Along with these, there is also a long list of tests that are run by your vehicle, the results of which can be seen within this response object. There are two Boolean properties for each test: 'available' tells you whether this test is available on your vehicle, and 'completed' tells you whether the test finished. So, for example, the following code looks at the oxygen sensor monitoring system.

```
if response.value.OXYGEN_SENSOR_MONITORING.
  available:
if response.value.OXYGEN_SENSOR_MONITORING.
  completed:
print("Oxygen sensor test completed")
```

Another special query and response involves looking at diagnostic trouble codes (DTCs). These are the codes that get set when you have an issue that turns on the check engine light on your vehicle. As a first step, the following code will give you the count of how

many DTCs were set.

```
response = my_car.query(obd.commands.GET_DTC)
print("There are " + len(response.value) + "
  trouble codes set")
```

The actual returned values are tuples, containing the actual error code, and a text description of the error code if it is one that the obd Python module understands.

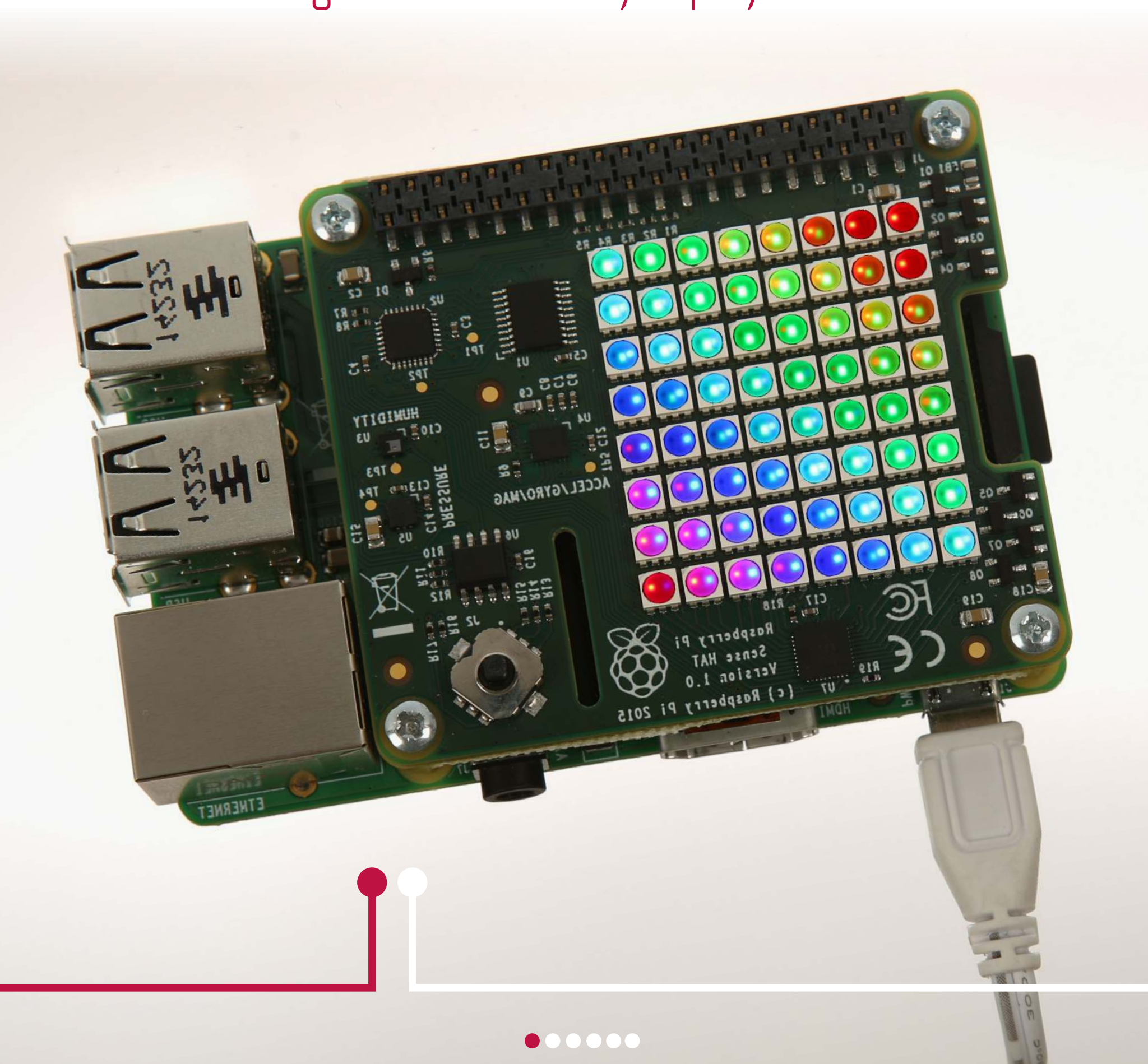
The one major problem, so far, is that the query method is a blocking function. This means that your program will be paused and have to wait until it returns before it can do anything else. This is not very user-friendly when trying to build a user interface to display all of this monitoring data. In these cases, you will be better served by using the asynchronous connection object instead. This new object uses a background thread to continually monitor the OBD connection and update the relevant values within the connection object. This means that any queries are returned immediately, since you don't need to go out and ask the engine. The following code keeps track of the engine's RPM value.

```
my_obd = obd.Async()
my_obd.watch(obd.commands.RPM)
my_obd.start()
print(my_obd.query(obd.commands.RPM)
```

When you are done, you can call the stop() method to stop watching the RPM value. Now you can have your own fancy car-monitoring system mounted to your dash.

Create a real-time LED humidity display with Python

Use the Astro Pi (aka SenseHAT) to take humidity readings and immediately display the results



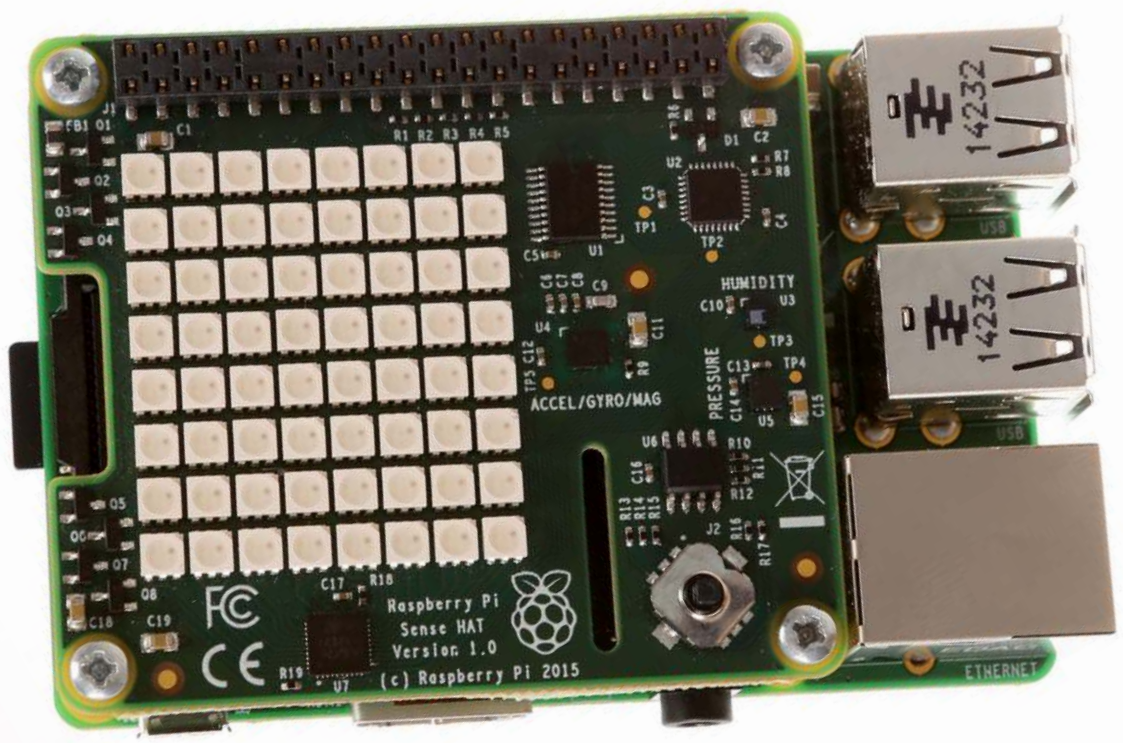
Humidity is a measure of how much moisture or water there is in the air. In this tutorial you will use Python to build a real-time humidity display. The program takes and stores a humidity reading using the SenseHAT's on-board sensor. Then it calculates a simple ratio to determine how many LEDs to turn on. In essence, the higher the humidity the more LEDs are turned on. Then the number of LEDs that need to be turned 'off' is calculated and these are added to the list. Finally, you'll set the program to display and 'turn on' the required number of LEDs. The program continually loops so both the reading and LED display is updated as the humidity changes.

01 Attach the SenseHAT

Ensuring that your Raspberry Pi is off, take the SenseHAT and attach it to the GPIO pins. Slot it firmly into place with the SenseHAT covering the main body of the Pi. Add the power supply and boot up the Pi. Open the LXTerminal Window and type sudo idle3 to open the Python 3 code editor. From the File menu, select New File to create a new program.

02 A test program

Next create a simple program to take a temperature reading and test that your SenseHAT is working correctly. The SenseHAT has a built-in heat sensor that can be used to read and return the current temperature. The sensor is close to the CPU and will pick up some of the residual heat. However, on the whole the reading is sound. Import the SenseHAT module and set the sense variable (lines 1 and 2). Take a temperature reading and



store it in a variable called temp (line 3). Finally, print out the current reading (line 4). Save your program and run it; you can do this by pressing F5 on the keyboard.

```
from sense_hat import SenseHat
sense = SenseHat()
  temp = sense.get_temperature()
print("Temperature: %s C" % temp)
```

03 Create the live humidity reader

Now that you have tested the SenseHAT, you can begin to create the main program. First, import the SenseHat library (line 1). Then use the clear code to turn off any LEDs that have previously been on (line 3). This ensures that the LED matrix is reset each time the program starts. Create two variables, one for the colour of the LEDs when they are on and another for when they are off. In this example the 'on' colour is set to red but you can change the values to change the colours to your own preference.

```
From sense_hat import SenseHat
sense = SenseHat()
sense.clear()
on_pix = [255,0,0]
off_pix = [0,0,0]
```

04 Take a humidity reading

Now it's time to take a humidity reading and round it to one decimal place. Begin by creating a while True loop (line 1), which means the program will continually take a reading and display the results. This ensures that the LED display is always updated and the readings are 'live'. On line 2 create a variable to store the humidity reading. The reading will be very accurate and contain several values after the decimal point. These are not required, so round up the reading to one decimal place (line 3).

```
while True:
    hum = sense.humidity
    hum = round(hum,1)
```

05 What about larger readings?

Depending on the time of year or where you are located, the humidity reading may be very high. However, above a certain value the readings become insignificant. Set a conditional to check if the reading is greater than 100. If it is then change the reading value to 100 (line one and two). This means that the top humidity reading is set to a maximum of 100 and makes the calculation of which LEDs to turn on easier. On the next line create a list called leds which will store the number of LEDs to turn on or

off. The SenseHAT has 64 LEDs so calculate the value of humidity which each LED represents (line four). (Each LED represents a value of 0.64 humidity.) Note that the first two lines are indented in line with the previous step.

```
    if hum > 100:
    hum = 100
leds = []
    ratio =  64 / 100.0
```

06 LEDs On or Off

You now have the components to work out the number of LEDs to turn on to represent the humidity. The formula ratio*hum takes the humidity reading and multiples it by the ratio from Step 5. For example, if the humidity reading is 50, then 50 multiplied by the ratio equals 32, half the LEDs. Convert this to an integer using int and store the value in a variable called on_count (line 1). To calculate the number of 'off' LEDs, subtract the number of 'on' LEDs from 64, since there are 64 LEDs in total (line 2).

```
on_count = int (ratio*hum)
off_count = 64 - on_count
```

07 Take a humidity reading

Now you have the total number of LEDs that need turning on or off. Write these values back to the list that you created in (Step 5), combining the colour and the total number of LEDs. First, take the colour of the 'on' pixels set in Step 1 and multiply this by the number of LEDs to

Take a different reading

Humidity measures the amount of water vapour there is in the air. You can replace humidity with 'sense. temperature' to create a simple digital thermometer. Put the SenseHAT in the fridge or near a warm heat source to take various readings. the image.

turn on. Then use the extend list function to write these values into the list created in Step 2. For example, if the humidity is 50, then the total 'on LEDs' will be 32 and this step will add 32 red-coloured LEDs to the led list.

```
leds.extend([on_pix]*on_count)
```

08 Add the 'off' pixels to the list and turn on the LEDs

In Step 6 you calculated the number of LEDs that needed to be turned off using the code: off_count = 64 - on_count. Use this value and multiply it by the colour that you set for the LEDs when they are off, in Step 1. (In this tutorial, (0, 0, 0).) Use the code, 'extend' to create a list which holds the 64 LEDs and record how many of these are 'on' and how many are 'off'. Now read the list and plot it onto the SenseHAT LED matrix using the code: sense.set_pixels(leds) line 2.

```
leds.extend([off_pix]*off_count)
sense.set_pixels(leds)
```

09 Run the program

That completes the program. Press F5 to save and run the program. Watch the LED display, which will display the current humidity as a ratio of the 64 LEDs. Try huffing onto the humidity sensor to see what happens. The number of LEDs displayed should increase. Challenge family and friends to see who can light up the most LEDs. off_count = 64 - on_count

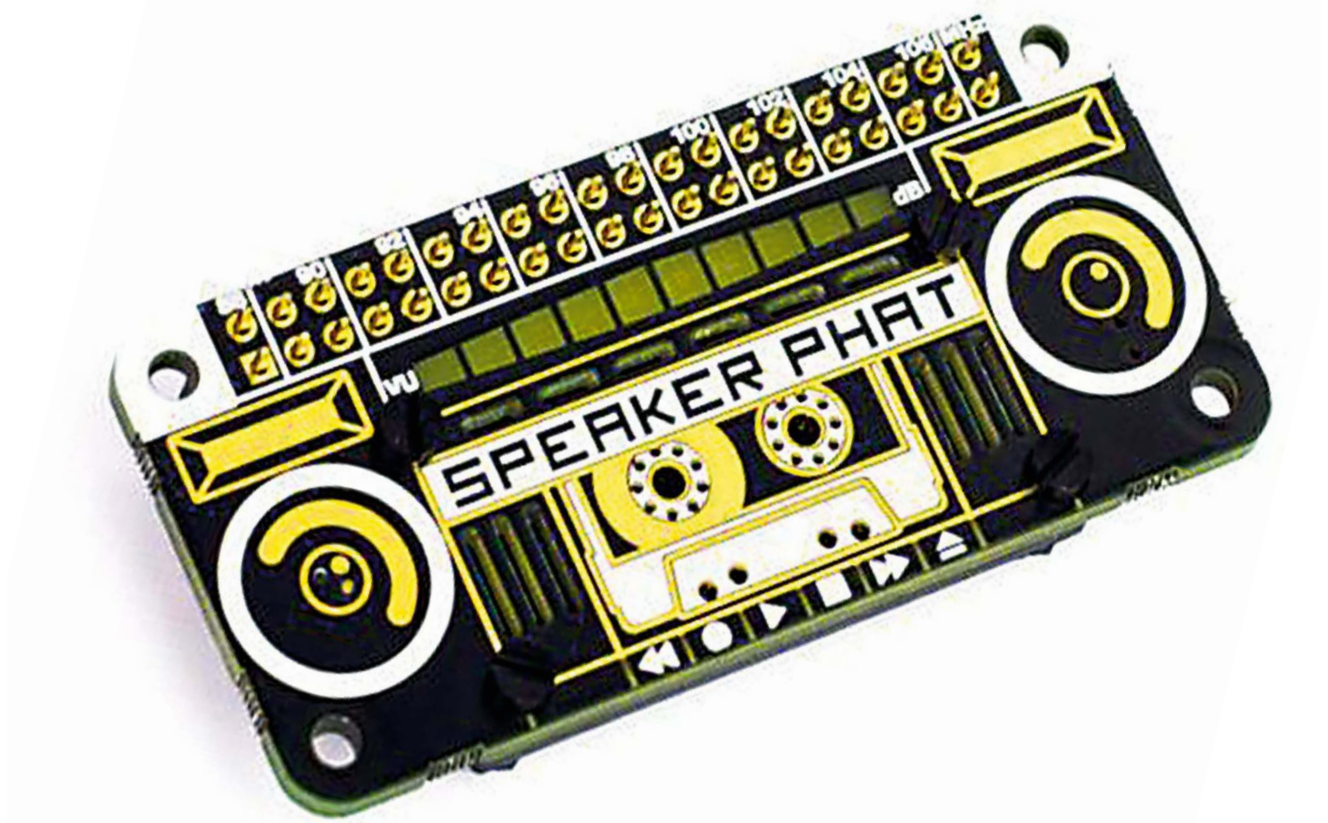
Red, Green and Blue colours

The RGB color model is an additive colour model in which red, green and blue light are combined to create a wide range of colors, over sixteen million different variations. The lowest value for a colour is zero which usually denotes black, as black is the absence of colour. The top value is 255. So red 255 means the maximum amount of red is being detected by the sensor.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Build a Zero MP3 player



Get this issue's source code at:

www.linuxuser.co.uk/raspicode